DOCUMENT RESUME

ED 093 703 SE 018 077

AUTHOR	Smith, Nancy Woodland
TITLE	A Question-Answering System for Elementary Mathematics.
INSTITUTION	Stanford Univ., Calif. Inst. for Mathematical Studies in Social Science.
SPONS AGENCY	National Science Foundation, Washington, D.C.
REPORT NO	NSF-EC-443X4
PUB DATE	19 Apr 74
NOTE	161p.; Psychology and Education Series, Technical Report No. 227
EDRS PRICE	MF-\$0.75 HC-\$7.80 PLUS POSTAGE
DESCRIPTORS	Computer Assisted Instruction; *Computers; Computer Science; Cybernetics; *Educational Technology; Elementary School Mathematics; *Language Research; *Linguistic Theory; Mathematical Linguistics; *Mathematics Education

ABSTRACT

This paper describes a project concerned with the understanding of natural language by computers. The project involves the development of both a theoretical model of natural language processing by computer and an actual implementation of the theory. The specific implementation chosen is a question-answering system for elementary mathematics which uses unrestricted natural language input. Details of the question-answering system are given and basic features in the perspective of the theoretical model are discussed. (JP)

ED 093703

BEST COPY AVAILABLE

A QUESTION-ANSWERING SYSTEM FOR ELEMENTARY MATHEMATICS

by

Nancy Woodland Smith

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

TECHNICAL REPORT NO. 227

April 19, 1974

PSYCHOLOGY AND EDUCATION SERIES

Reproduction in Whole or in Part Is Permitted for
Any Purpose of the United States Government

SE 019 077

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES
STANFORD UNIVERSITY
STANFORD, CALIFORNIA

Table of Contents

Chapter

Section

Page

Acknowledgments v

I. Introduction 1

I.1 General Introduction 1

I.2 Basic Components of the System 2

I.3 Choice of Subject Matter 2

II. The Theoretical Model 10

II.1 Comparison with Other Systems 10

II.2 Transformations 38

II.3 Restructuring 44

III. CONSTRUCT and the Grammar 51

III.1 CONSTRUCT 51

III.2 The Scanner and the Dictionary 53

III.3 The TRANSL File 57

III.4 The Grammar 59

IV. The Rules of the Grammar and their Semantic Functions . . 70

IV.1 Introduction 70

IV.2 S-Rules 76

IV.3 F-Rules 77

IV.4 Top-Level EXP-Rules 78

IV.5 Types of EXP's 80

IV.6 EXP1-Rules 81

IV.7 Set-Expressions and Ntuples 83

IV.8 DATEXP and TIMEXP-Rules 85

IV.9 ARITHEXP-Rules 85

IV.10 UNIT and NUNIT-Rules 86

IV.11 Geometric Measurements 88

IV.12 Relative Clauses 89

IV.13 Prepositions 91

IV.14 SUBST-Rules 97

IV.15 Arithmetic Relations 100

IV.16 Adjective Rules 101

IV.17 CONVUNITS-Rules 101

IV.18 CONVPREP-Rules 102

IV.19 SPECPREPHRASE-Rules 103

IV.20 SPECPREP1-Rules 103

IV.21 ORDERING-Rules 105

IV.22 Commands Using Special Verbs 105

IV.23 Arithmetic Commands 106

IV.24 Basic Command Rule 107

IV.25 Special Conversion Commands 107

IV.26 Combinations of Commands 108

IV.27 Declaratives 109

IV.28 NP-Rules 110

IV.29 NP1-Rules 112
IV.30 NP2-Rules 114
IV.31 NP3-Rules 116
IV.32 NP4-Rules for Set Nouns 117
IV.33 NP4-Rules for Function Nouns 118
IV.34 2FCN-Rules 121
IV.35 Existence Questions 122
IV.36 If Questions 122
IV.37 Idiomatic Question Formats 123
IV.38 Questions With Introductory Clauses 123
IV.39 Questions Beginning with a Linking Verb 124
IV.40 Questions Beginning with an Auxiliary Verb 125
IV.41 CHOICELIST Questions 127
IV.42 Q1-Rules 129
IV.43 HOWMANY Questions Involving UNITs and NUNITs 129
IV.44 Other HOWMANY Questions 131
IV.45 Interrogative Questions 132
IV.46 FCNHNP-Rules 133
IV.47 HNPAS-Rules 133
IV.48 COMP1HNP and COMP2HNP-Rules 134
IV.49 HAVENPF-Rules 135
IV.50 HAVENP-Rules 136

Appendix I

Examples of Questions and their Answers 138

Index 147

References 149

Acknowledgments

I wish to express my deep gratitude to my husband, Dr. Robert L. Smith, Jr. for all his help with the project. He also deserves very special thanks for the typing of this dissertation, the many hours of babysitting with our daughter, and the large amount of advice and encouragement that he provided me at all stages of the undertaking.

I would also like to thank Dr. Freeman L. Rawson, III for his contribution to the question-answering system, Professor Patrick Suppes for serving as my advisor and for providing the computer facilities for this project, and Professors Dov Gabbay and J.M.E. Moravscik for participating on my reading committee.

This research was supported by National Science Foundation Grant EC-443X4.

BEST COPY AVAILABLE

Chapter I

Introduction

I.1 General Introduction

This paper describes a project concerned with the understanding of natural language by computers. The project involves the development of both a theoretical model of natural language processing by computer and an actual implementation of the theory. The specific implementation that we have chosen is a question-answering system for elementary mathematics which uses unrestricted natural language input.

A complete explication of the theoretical issues can be found in [22] and additional information on the project is also given in [17]. This paper is primarily concerned with describing the question-answering system and then discussing its basic features in the perspective of the theoretical model.

In this chapter, I will give a general description of the operation of the question-answerer and then discuss our reasons for choosing this particular implementation of our theory. Chapter II includes a discussion of the theory, a comparison with other systems, and a section on transformations. Chapter III gives a more detailed discussion of the components of the system and the final chapter contains a listing of all the syntactic rules with their associated semantic functions and a few brief comments on each group of rules. The APPENDIX contains examples of questions currently answered by the question-answering system.

I.2 Basic Components of the System

There are five basic components of the system. 1) CONSTRUCT is a SAIL program which provides the interface between the components and handles the actual parsing. 2) The Scanner which is a part of CONSTRUCT preprocesses the input using both a dictionary of lexical categories and a file, called the TRANSL file, of strings of words that require special preprocessing. 3) The grammar is a context-free grammar (cfg) read into the program at runtime. 4) Each rule of the grammar has an associated semantic function whose explicit arguments are the meanings of the elements on the right-hand-side of the rule. The function when evaluated returns the meaning of the left-hand-side. 5) The result of the semantic parse which is called the semantic construction is passed to the Evaluator which is programmed in LISP. It evaluates the semantic construction and returns the answer.

I.3 Choice of Subject Matter

Our decision to implement the ideas we had about natural language processing in a question-answering system had several motivations. First, the question-answering format provides a thorough work-out for all the components of the system and also produces hard results by which the correctness of the various components can be judged. In order to answer a question correctly each part of the system must perform its job well. First the analysis by the syntactic and semantic components must correctly determine the meaning structure of the question; and then the evaluation routines together with the data base must produce the answer based on the meaning structure provided by the natural language processing components. If a system does not implement question-answering, its analyses of individual sentences may appear to be intuitively plausible and the data base to be well-integrated, while in fact, the analyses may not be detecting all the subtleties of meaning and the data base may not include all the proper inter-relationships. For example, systems which on the surface appear to be giving correct analyses of input sentences may not be able to support adequately such constructions as quantification, inference, or belief structures. Of course, a close theoretical study of a given system will reveal its capacities and limitations regarding these sorts of constructions and systems should be so scrutinized, but implementing question-answering provides an additional objective practical way of gauging a system's power.

A motivation very similar to ours is given by Woods for his airline schedule question-answerer in [27]:

> The objective of the research described here has been to develop a uniform framework for performing the semantic interpretation of English sentences. It was motivated by the fact that,

although there exists a variety of formal parsing algorithms for computing the syntactic structure of sentences, the problem of using this information to compute their semantic content remains obscure. A question-answering system provides an excellent vehicle for such a study, because it forces consideration of semantics from the point of view of setting up correspondence between the structures of a sentence and objects in some model of the world (i.e., the contents of the data base).

Another obvious motivation for choosing a question-answering system lies in the ultimate practicality of a working question-answerer especially in our chosen subject area of elementary mathematics. And a third crucial motivation is the desire to extend our efforts from the analysis of natural language to the generation of natural language. This has not yet been implemented, but the system that we have developed thus far provides a good basis for the task of generating natural language answers to questions.

Our next necessary choice was the subject matter for the question-answerer. Various subjects were considered and five main aspects of each were evaluated:

1) The subject matter itself and how it could be represented and dealt with as a data base;

2) The fragment of natural language commonly used to pose questions and state facts about the subject;

3) The type of questions most commonly asked and their amenability to computer-answering;

4) The ease of extendability of the finished system to other subject matters;

5) And finally, the potential use that might be made of a question-answerer in the area.

Elementary mathematics is a good choice in each of these respects. The subject matter is well-defined and easily represented as a data base. A wide range of questions can be answered without requiring a large number of facts in the data base. The need for massive factual data was our basic reason for rejecting such topics as geography which are suitable in other respects. Minsky discusses the reason why mathematics is so often chosen as subject matter in [10].

> It is not that games and mathematical problems are chosen because they are clear and simple; rather it is that they give us, for the smallest initial structures, the greatest complexity, so that one can engage some really formidable situations after a relatively minimal diversion into programming.

Elementary mathematical data with the exception of some tabular information used for unit conversions, etc. is largely procedural. We use two basic data types, sets and functions. This division cuts across the boundaries of traditional parts of speech. Verbs like 'add' and 'multiply', adjectives such as the comparative adjectives, and nouns like 'factor' and 'area' are all represented as functions. Other adjectives such as 'even' and 'prime' and nouns like 'number' and 'fraction' are dealt with as constructive sets which are represented by characteristic functions. This means that every mathematically substantive word in the vocabulary (with the exception of those related to tabular information) will be represented in the data base as a function, either a primitive mathematical function which can be applied to its argument(s) or the characteristic function of a set. (See Chapter II for a discussion of other data types which can be used if the subject matter requires them.) To handle mathematically (although not necessarily grammatically) simple questions, it is not necessary to store any information about the inter-relationships among these functions or any composite functions. The semantic component handles the various combinations. For example, it is not necessary to have an EVENFACTOR function. Consider the following two questions:

1) Is 2 an even factor of 6?
2) Does 6 have any even factors?

In the first case, the FACTOR function is applied to 6 and the result is the set {1,2,3,6} which is intersected with the set of even numbers by the semantic function for intersection yielding the set {2,6}; then the semantic function for subset checks if {2} is a subset of {2,6}.

To answer the second question a transformational semantic function is used. The argument to the FACTOR function, 6, is not contained in the noun phrase and is inaccessible to it at the NP-level. So a transformational semantic function creates an EVENFACTOR function for use at the higher level. (For details of this type of semantic function see Section II.3). Note that the EVENFACTOR function does not need to be permanently stored in the data base rather it is created at runtime. In order to handle more mathematically complex questions, some heuristic information about the intersections of various sets will be needed. For example, the system is now programmed to know that the intersection of 'even' and 'prime' is the singleton set {2}.

Another desirable feature of elementary mathematics is that the subject matter is self-contained. Previous implementations of this approach to semantics [21] involved the analysis of corpora of child utterances. It was discovered that doing any real work with the semantics would have necessitated the building of a model of the child's interactions with her environment. The decision was made that more intense study of the natural language itself should be the thrust of the investigation at this stage. So we have chosen a project that does not involve modeling of an individual's interactions with the world. However the closely related problem of dealing with the context of the complete dialogue with the computer cannot be avoided by choice of subject matter. There is always the possibility in a question-answerer that one question will be related to a previous question or answer by an anaphoric reference. Again, while recognizing this as an extremely important problem, we have decided that it is not a suitable problem for our first stage of development. A survey of questions in elementary textbooks proved that in fact we could compile a large sample set of mathematical questions which were independent of their context. Note that this does not imply that our semantic functions will have more than the usual difficulties with these constructions which are a problem for any system. Preliminary work has shown that we will be able to write the appropriate semantic functions for context-checking. It is simply a matter of choosing a manageable set of problems for the initial development of the system.

The fragment of natural language used to talk about elementary mathematics does contain all the traditional parts of speech and all the varied sentence formats. Also, the vocabulary is limited enough to be manageable but sufficiently rich to cover many linguistically interesting constructions. We found very few grammatical constructions that were peculiar to this subject matter. This means that to change or extend the scope of the question-answerer will require extension rather than replacement of the current grammar. For example, a large part of our efforts have been devoted to prepositions and sentences using the verb 'to have'. Both of these are surely problems common to all substantial fragments of natural language. The fact that not all senses of the verb 'to have' and only fourteen of the prepositions were found to occur in an elementary text on mathematics [23] gives us a workable starting point for these constructions.

It is desirable to have objective sources of sample questions so that a wide range of sentence formats will be included. There are two readily available sources of elementary mathematics questions. During the developmental stages, a good source of questions is elementary textbooks. However, now that we have a working model, we plan to develop a CAI program using the question-answerer so that we can gather sample questions from elementary students. We expect the new questions to be less standardized than those from the textbooks with respect to vocabulary, grammar, and subject matter. This will provide raw data for the second stage of the project in which we will be concentrating on such major problems as anaphoric reference, habitability, learning and ambiguity.

Also, we expect that testing the system with elementary students will confirm our hypothesis that elementary mathematics is a suitable subject for a practical question-answerer. We conducted an experiment with elementary students in which we simulated a question-answerer for Black History and the results were discouraging. The questions asked, in general, called for value judgments and causal explanations that were well beyond the range of current work in artificial intelligence.

Chapter II

The Theoretical Model

II.1 Comparison with Other Systems

Early programs for natural language processing were concerned primarily with syntax. The current trend is to place the primary emphasis on semantics. Our major interest is in clarifying the relationship between syntax and semantics. This issue is discussed by Katz in [8].

> ...the semantic competence of a speaker enables him to obtain the meaning of new sentences, and other new compound syntactic constituents, as a compositional function of the meanings of their parts and grammatical relations. Since infinitely many possible sentences are novel arrangements of familiar lexical items, this assumes that the speaker's semantic competence provides him with meanings for each of the finitely many lexical items of his language and a set of rules for combining the meanings of linguistic constructions to compositionally form the meaning of each sentence of his language and each compound constituent of each sentence.

Winograd [26] comments that "Often the most important clues about what is being said are the syntactic clues." These "clues" form the basis of our semantic functions. Each syntactic construction which is represented by a rule in the grammar has its own semantic function that shows how to obtain the meaning of the construction from the meanings of its parts. It is necessary to make a distinction not made by Katz in the above passage between lexical items which have their own "meaning" and lexical items such as determiners, auxiliaries, relative pronouns, etc., which function like the syntactic structure as a whole to give guidance as to how the meaningful elements are to be combined. For example consider the following question and the rule which parses it:

```
EX1:    How many factors of 12 are even numbers?
RULE1:  Q <- /HOWMANY/ NP LINK NP   (CARDINALITY (I ;2; ;4;))
```

[Note: CARDINALITY and I (intersection) are primitive semantic functions.]

The numbers enclosed in semi-colons in the semantic functions refer to the position of the elements on the right-hand-side of the rule. The two NP's will be parsed at a lower level and the semantic functions for them inserted in the proper position before the complete semantic construction for the question is passed to the evaluator. The important point to note is that each question which has this basic syntactic form can be answered by intersecting the two sets and finding the cardinality of the intersection.

The terminology here is perhaps misleading. The name 'semantic function' can refer to one of three things depending on the context. We often refer to the primitive semantic functions such as CARDINALITY and I as simply semantic functions. We also speak of each grammatical rule as having an associated semantic function such as (CARDINALITY (I ;2; ;4;)) which was given above. And finally, each sentence parsed will have its own final semantic function usually called the semantic construction which is passed to the evaluator. For example, in this case the semantic construction will be:

(QUS (CARDINALITY (I (APP @FACTOR (LST 12)) (I @EVEN @NUMBER)))).

> [Note: QUS is the semantic function used to indicate that the input was a question. APP is the semantic function used for applying functions to their arguments, in this case, the FACTOR function to 12.]

This semantic construction shows us how the four meaningful words in the original sentence can be combined to find the meaning of the entire sentence.

There are two basic types of primitive semantic functions. The first type are the substantive semantic functions. Many of these are standard set-theoretical functions like cardinality, intersection, union, set difference, and set complement. There are also functions for dealing with comparatives and ordinals. The function APP applies mathematical functions like FACTOR that are found in the original sentence to their designated arguments. A function called EXIST checks whether or not a set is empty. The function ENMF checks the cardinality of a set against a given number and is used for constructions like 'the 6 factors of 12'. There are also semantic functions which are designed specifically for this subject matter. These include functions for each of the basic arithmetic operations and special functions for dealing with mixed numbers, percents, expressions of units of measurement, etc. Many more of these various sorts of substantive semantic functions will be discussed in the examples in the following chapters.

The other type of primitive semantic functions are used to establish a control structure for the evaluation phase. We generally refer to these functions as transformational semantic functions because they, in our system, deal with the constructions which are often viewed as more complicated transformations of simple constructions. The transformed construction differs from the "kernel" by having its elements out of the standard order and/or having some of its elements suppressed. There are of course other possible related features of the transformed construction such as a change of voice from active to passive or a change of verb from 'is' to 'have'. I will discuss the details of our handling of the various kinds of transformations in Section II.2. The basic problem with handling non-standard word orders is that the evaluator which is written in LISP uses recursive inside-out evaluation. Therefore without the transformational semantic functions to provide a control structure the evaluations would be made in the wrong order. This is particularly obvious in questions like

Does 6 have a factor of 2?

which is a "transformation" of the question

Is 2 a factor of 6?

Special semantic functions are used for noun phrases appearing with the verb 'to have'. Part of their job is to ensure that the evaluator will not attempt to apply the function, which in this case is the FACTOR function, at the innermost level because its argument is in fact somewhere else in the sentence. Through the use of these functions transformations can be handled without sacrificing the recursive inside-out character of the evaluator.

I will first discuss the advantages of our approach to some of the common problems of natural language processing; and then I will discuss specific criticisms that have been levelled against the use of context-free grammars for natural language processing and show how the addition of semantic functions enables us to overcome the customary problems with cfg.

This method of approach to the construction of a natural language understanding and generation system has advantages in three main areas: clarity, flexibility, and extendability. As mentioned above, our major interest is in clarifying the relationship between syntax and semantics. This does not mean that we believe there are two sharply defined, distinct, and independent elements of language which have traditionally been called "syntax" and "semantics". It is clear that the two work together. Some systems such as those of Winograd and Woods, as a result of recognizing this interaction, have eliminated distinct phases of analysis corresponding to syntax and semantics. Instead syntactic and semantic routines can call each other and the results determine how the analysis will proceed. The disadvantage of this total interaction is a loss in clarity. The actual mechanisms of interaction may be buried deep in the program. The basic structure of our system can be understood without examining any of the specific programs. The interaction between the syntactic and semantic features of the language is captured in several ways.

First, rather than writing an adequate grammar for the language we are dealing with and then imposing a semantics on the grammar, we have instead developed a fairly nonstandard grammar which is completely responsive to the semantic needs of the analysis. Each rule has been written with a clear idea of which semantic function will be used and consequently which elements of the input sentence need to be parsed at that level for use as arguments to the semantic function. For example, noun phrases containing the preposition 'of' like 'factors of 6' and 'denominator of 1/2' should not be parsed as a noun followed by a prepositional phrase. The important semantic insight about these phrases is that they contain the name of a function which is stored in the data base and the argument to the function both of which are needed as arguments to the APP (apply) semantic function. Therefore we might have the rule:

RULE1: NP <- FCN /OF/ NP (APP ;1; ;3;) .

[Note1: The actual rule is more complicated to account for modifiers and lists of function names or arguments.]

[Note2: /OF/ is the lexical category for the preposition 'of'.]

However, certain other prepositions in this position can be parsed by the rule:

RULE2: NP <- N PREPHRASE (I ;1; ;2;)

[Note: The category FCN is used for nouns which name functions and the category N for nouns which name sets.]

Examples of this type of preposition are 'between' (ex. '4 is between 3 and 5'), 'in' (ex. '8 is in the set {7,8}'), and 'before' and 'after' (ex. '6 comes before 7'). These prepositional phrases can occur in several grammatical positions in the sentence, but in each case the meaning of the prepositional phrase itself can be determined regardless of the context. The result of evaluating each of these prepositional phrases will be a set. Thus to handle the noun phrase, 'the prime between 6 and 10', we can use RULE2 which will intersect the set of primes with the set of numbers between 6 and 10. It is not necessary to spell out at the level of RULE2 which particular preposition in this category will be used. The intersection function will have the sets it needs passed up to it from a lower level. In RULE1, the APP function does explicitly need to see that the preposition is 'of' and it needs both the noun phrase before and the noun phrase after the preposition to use as arguments.

This recognition that all the relevant arguments to a function need to be parsed at the same level has lead to a grammar with much flatter trees than are usually associated with context-free grammars used for natural language processing. For example, this grammar does not contain the standard rule:

S <- NP VP.

It would be extremely difficult to write a good semantic function for this rule. It is necessary to know more about the verb in order to determine which semantic function is needed, and moreover, if there is a noun phrase in the VP, the verb will be determining the relationship between the two NP's so they will both be needed as arguments to the semantic function established by the verb and will both need to be parsed explicitly at the same level.

A second way in which our system is able to have separate syntactic and semantic components which act in parallel rather than interactively at runtime is by not holding the traditional conceptions of syntax and semantics sacred when making the decision as to which component will handle any given aspect of the language. For example, we have not as yet needed to implement routines for checking agreement, but they will be implemented as semantic functions rather than as part of the grammar since a cfg cannot deal with agreement satisfactorily. The semantic functions also handle transformations which is a traditional syntactic function. The primary way in which the grammar incorporates traditionally semantic features is through the use of "semantic categories" rather than the standard lexical categories.

There have been several types of semantic categories used in recent years. Katz [8] has proposed that the dictionary entries for words should contain "lexical readings" which include the various "senses" of the word, for example, a given noun might be a physical object, inanimate, etc. This information can then be used for disambiguation. A combination of "selection restrictions" and "projection rules" eliminates those interpretations of sentences which are based on inconsistent "senses". For example, Katz gives the following reason for the rejection of the incorrect interpretation of the sentence:

(1) The man hit the colorful ball.

> . . . (1) has no meaningful interpretation on which 'ball' has the sense of a social activity, even though it has this as one of its senses in the dictionary, because of the conceptual incongruity of relating a social activity to a physical action such as hitting by making it the object on which the action is performed.

There have been various objections to this approach. Palme [12] objects on two grounds. First he points out that it cannot handle disambiguation of sentences like "He went to the park with the girl." which require contextual information for disambiguation. Katz, in fact, states that this aspect of his system is not intended to handle this type of ambiguity. Palme's second objection may be more serious. He believes that the very large complex dictionary needed for this sort of system will duplicate information which needs to be in the data base. This objection can only be evaluated on the basis of a particular implementation although it does seem, as Palme claims, to be more natural to have all the information unified in a single data base. Minsky [10] also believes that these semantic categories, which Katz calls "lexical readings", are not truly "grammatical" categories and that the information which they convey should instead be included in the form of a world model in the data base. Minsky's argument is that while relations taken one at a time could be handled these multiple categories lead to interacting relations which require a very powerful logic.

Katz' approach is the most common way of introducing semantic information at the level of the dictionary, but it is not the approach that we have taken. Our approach is rather a combination of two other methods which have been used in recent systems. One method is for functional words and the other for the substantive words. As noted above, the grammar has been written to facilitate the writing of the appropriate semantic functions. In line with this objective, each functional word has its own lexical category assigned to it in the dictionary. Thus, at the point that the word is parsed, the semantic procedure associated with the functional word can either be applied immediately as is the case in DEACON [24] [5], which also gives each functional word its own lexical category, or encoded in the semantic construction for the sentence as is done in our system. An example is the preposition 'of'. The semantic function in our system for 'of' is APP. The rule is:

```
        NP <- FCN /OF/ NP                (APP ;1; ;3;) .
```

So the semantic construction for 'factors of 6' will be

```
        (APP @FACTOR (LST 6)).
```

The data in the DEACON system are stored in ring structures. The semantic procedure applied when 'of' is parsed is to substitute for the whole phrase the third member of the ring containing both the noun preceding and the noun following the 'of' in the input. For example, if the input is "commander of the 638th battalion", procedures will first be used to eliminate the determiner and also to eliminate the word 'battalion' as being redundant, thus leaving "commander of 638th".

In the data base, there will be a ring connecting 'commander', '638th', and 'Jonathan M. Parker'. The rule for 'of' will substitute 'Jonathan M. Parker' for 'commander of 638th'.

Our handling of the classification of substantive words is based on insights very similar to those of Sager [18].

> The discourse in a science subfield has a more restricted grammar and far less ambiguity than has the language as a whole. We have found that the research papers in a given science subfield display such regularities of occurrence over and above those of the language as a whole that it is possible to write a grammar of the language used in the subfield, and that this specialized grammar closely reflects the informational structure of discourse in the subfield. We use the term sublanguage for that part of the whole language which can be described by such a specialized grammar.
>
> The sublanguage grammar provides a method for developing the particular word classes (the special-word sets) and the relations among these classes which are of special significance in a given science subfield, i.e., which are the linguistic carriers of the specific knowledge in the subfield. Yet these categories and relations are not determined a priori for the subfield. Rather, they are the interpretation of the formal grammatical categories and relations of the sublanguage grammar. Thus, in the pharmacological sublanguage which was investigated, the two noun subclasses I (containing, e.g. ion, K+) and G (containing, e.g., drug, digitalis, glycosides), which in the subfield have the significance "ions" and "pharmacological agents," respectively, and play crucially different roles in the physiological mechanisms being described, are obtained as separate classes because they occur with different classes of verbs: e.g., I as the object of such verbs as transport, G as the subject of such verbs as inhibit. It then turns out that the sublanguage word classes, which are established on the grounds of what other grammatical classes they occur with

> (as subject, object, etc.), are the linguistic counterparts of the real-world objects, events, and relations which are studied and described in the given subfield.

While the phenomenon she describes is certainly more pronounced in subfields, we have found that it does occur in natural language as a whole. Sager has performed several analyses which we have not that have very interesting results. She claims that as a result of either string decomposition, transformational decomposition, or a transformational lattice, the three kinds of vocabulary appear in three distinguishable portions of the decomposition. The bottom nodes contain the specific vocabulary of the subfield, the intermediate nodes the general scientific vocabulary and the top nodes the vocabulary expressing "the scientist's conclusions, doubts, speculations, etc." She obtains another interesting result by comparing investigations of current articles in the same scientific field performed at different times. The discovery was that certain words which were new to the vocabulary at the time of the initial study functioned as operators on elementary sentences at that time but later were found increasingly as subjects of new elementary sentence types. Thus the evolution of the grammar parallels the advance of the science, or as Sager puts it, is a "representation" of the advance.

Our primary goal is to find how the grammar is related to the meaning. Both our system and Sager's use the idea that categories can be formed which contain words that are both naturally related to each other with regard to the subject matter and naturally related with regard to their grammatical role. This differs from Katz' approach which is primarily concerned only with semantics. His semantic categories are not the grammatical categories but rather are included in the dictionary in addition to the grammatical categories. Because the words in his dictionary have multiple semantic categories it would be very difficult to incorporate them into the actual grammar. Also, while certain of his "senses" such as the distinction between mass and count nouns have a grammatical counterpart, in general unless all the categories are built into an extremely sensitive grammar, there will be no grammatical difference between two semantic interpretations of a given sentence. For example, there will be only one parse of

The man hit the colorful ball.

The disambiguation of this sentence is properly part of the semantic component in his system and in ours it will be part of the evaluation. There is no obvious way to expect help from the grammar in disambiguating the sentence. We are, however, concerned with finding those areas where the grammar and semantics can help each other. Our purpose for using nonstandard lexical categories is not to aid in disambiguation but rather to aid in developing a grammar which will produce the most correct parses with respect to the meaning of the sentence. Of course, as a natural by-product, this carefully worked out grammar will eliminate a large amount of unnecessary grammatical ambiguity. For example, our system divides nouns into two basic categories N (nouns that name sets) and FCN (nouns that name functions). Given a list of noun phrases some of which may themselves be composed of lists and at least one of which contains the preposition 'of', without the distinction between N's and FCN's there will be ambiguous parsings. The presence of 'of' indicates that there is a function name or list of names and the argument(s) to the function(s). Without the category FCN to pinpoint the function name(s) in the list of noun phrases, the rules would be

```
NP <- LISTOFNP
NP <- NP /OF/ NP
```

thus allowing the grammar to parse very strange lists of noun phrases by choosing incorrect sublists to fill the two slots of function name and argument. This shows how the grammar and semantics can help each other. The two types of nouns must be evaluated differently. Identifying the type at the syntactic level thus provides very useful information to the evaluator through the semantic function. Historically in our system the distinction was discovered while writing the evaluation routines. However, the distinction is also a very important one in the grammar. The two types of nouns in isolation never have the same grammatical role although a noun phrase which contains an FCN and its argument like 'factors of 6' will evaluate to a set and therefore can be used in the same position as an N except that it will never function as an appositive noun. Thus the distinction is necessary in the grammar to prevent senseless ambiguities.

Woods in [27] points out the same distinction when he discusses functional and non-functional noun phrases. However, his airline schedule program does not utilize the grammatical features of these phrases to determine the semantics. He lists seven of the N-rules for functional noun phrases and one sample noun phrase for each. For example,

```
N6      1-(G8:(1) = (departure time) and
        2-(G10:(1) = of and FLIGHT ((2))) and
        3-(G10:(1) = from and PLACE ((2)))
        => DTIME (2-2,3-2):
```

e.g., "the departure time of AA-57 from Boston".

The entire seven sample sentences for the rules are:

N6 The departure time of AA-57 from Boston
N7 The arrival time of AA-57 in Chicago
N8 The operator of AA-57
N9 The time zone of Boston
N10 The number of stops of AA-57 between Boston and Chicago
N11 The type of plane of AA-57
N12 One-way first-class fare from Boston to Chicago.

This indicates that each such noun-phrase has its own N-rule in his system. In N10, Woods treats "number of stops" as a single function. Considering that both "number of" and "stops of" can be used independently of each other, I believe that these examples contain eight rather than seven function names. The examples do bring up a problem that we have not had with our subject matter. At least some of the function nouns in these examples can be used in other contexts as set nouns. For example, time zone can be a function taking the name of a place as argument and returning the time zone of the place, but time zone can also be viewed as a set containing the names of all the time zones as in 'List all the time zones!' To include these nouns in our system would necessitate giving them the multiple lexical category of N&FCN, but would probably not lead to grammatical problems or to problems in the evaluator if both representations, as a function and as a set, were stored and the correct one selected on the basis of the parse. However, a more serious problem would arise with questions like 'What are the time zones in the United States?' This clearly is a function and yet our current grammar would parse it only incorrectly as a set noun. The semantics of the sentence is clear which indicates that our grammar does not yet include all the grammatical formats associated with function nouns. All of the nouns which can only be function nouns are found only with the preposition 'of' or in one of several formats used when the main verb is 'have' (see Section II.3). However preliminary work in the area of time and calendar-type problems shows that there is an area of elementary mathematics which does use nouns that have both representations and there are more grammatical options associated with them. For example,

a) Which month comes after March? -- set
b) What is the number of months in a year? -- function
c) What are the months of the year? -- function

A preliminary guess would be that these nouns when used as function names can use either the preposition 'in' or 'of'. It is interesting, although not surprising since there is also a set representation, to note that these functions seem to all be implementable as table lookup procedures. This indicates another correspondence between the grammatical structures and the subject matter.

By writing a separate rule for each functional noun phrase found in his subject matter, as it appears he has done, Woods fails to utilize the common grammatical features of these nouns phrases which indicate the similarity of semantical treatment. Not only is this inefficient but it also necessitates the writing of new rules as the subject matter is extended. In our system, new function nouns need only to be added to the data base and the dictionary. Four of the eight examples that Woods gives could be handled by our current rule:

NP <- FCN /OF/ NP (APP ;1; ;3;)

These are: operator, time zone, number, and type of plane. The other four examples contain an additional prepositional phrase which gives the PLACE(s). A grammatical slot would need to be added to our rule to account for this and probably some other modification made to prevent ambiguous parsing of the prepositional phrase.

The use of the two categories N and FCN is extremely useful both to the precision and clarity of the grammar and to the correct writing of the semantic functions. The semantic categories used by Katz focus on the individual words in the vocabulary. Our system instead focuses on each of the categories like noun, verb, adjective, etc., and tries to discover general patterns within the use of that category which form the basis for interesting grammatical and correlative semantic distinctions. Since Katz' semantic categories operate on individual words and the words in a group like the noun group may have overlapping categories, his system has little utility as a grammatical device; it is intended to serve in the semantic phase of analysis. There are undoubtedly large numbers of sentences like "The man hit the colorful ball." which must be disambiguated in the evaluation phase on the grounds that one of the interpretations makes no sense. (See Section III.4 for a further discussion of ambiguity). However before a given possible interpretation can be ruled as either meaningless or meaningful but not as appropriate in the context as another possibility, the set of possibilities must be generated. One useful way of looking at the difference in emphasis between our system and other systems is to say that many recent systems have concentrated their efforts on the analysis of the possible interpretations while our primary emphasis is on their generation.

Winograd [26] discusses three models of semantics: categorization model, association model, and procedural model. Our system like his falls under the procedural category. Both the categorization and association models make relatively little use of syntax. The categorization model is based on the semantic categories of Katz and Fodor and is used in systems like Schank's conceptual parser [20]. Schank has extended the semantic category system to include for each sense of a word how that word relates to other words, for example, whether or not it takes an object and if so what category the object must be. The association model which is used by Quillian [14] [15] stores the content words in the vocabulary in a network with links between the words to represent their associations in the subject matter. The meanings of phrases are found by finding the links between the content words of the phrase. A third method not mentioned by Winograd is the pattern recognition method. This was used in the ELIZA program [25] and more recently by Colby et al. [4]. Here the input is scanned for certain key phrases. The assumption being that a large part of natural language is merely fillers and unimportant idiomatic phrases. Each of these three methods concentrates primarily on the semantics. The semantic routines simply pick out anything they might need from a rough parse; there is no deep systematic grammatical analysis done. Winograd characterizes this attitude by saying:

> There is also a complexity of syntactic parsing. The semantic connections might give clues to the underlying structure which would change the parsing task into simply checking the plausibility of the relations, and cleaning up the details. This is the approach taken by both Schank and Quillian.

We view the role of syntax as much more important than this. Each grammatical structure indicates the semantic procedure necessary to evaluate its meaning. Based on the grammatical parse of a sentence, a semantic construction is assigned to it. The purpose of the semantic construction is to give the evaluator the necessary functions and the control structure for applying them.. It is at the level of the evaluator, which has access to the data base including the meanings of words and the context of the conversation, that semantic disambiguation can be done if necessary.

It is because of this close relationship between the grammar and the semantics that the lexical categories must be carefully chosen so as to maximize the information that can be obtained from the parse. The basic categories for nouns are N, FCN, and 2FCN. The category 2FCN is for function nouns like 'intersection' and 'sum' which always take more than one argument. In addition there are noun categories which are analogous to FLIGHT and PLACE which were used in the example from Woods' system. These are categories which add both clarity and precision to the grammar. Examples in our system are: GEOFIGURE for the names of geometric figures, i.e., 'rectangle'; UNITS for units of measurement like 'ounce', 'foot', 'tablespoon', etc.; NUNITS for 'ones', 'fives', and 'tens', etc. which are evaluated differently than regular units; and /3D/ for nouns like 'length', 'height', and 'width'. These categories are probably not necessary to prevent grammatical ambiguity but for developmental purposes they make the grammar much more readable and specific semantic functions can be assigned to the rules before the more general procedures are discovered.

The adjective categories are: ADJ for regular adjectives like 'even', 'finite', and 'prime'; /OPER/ for 'square', 'cubic', etc..; ORDADJ for ordinal adjectives like 'first' and 'second'; COMPADJ for the comparative adjectives like 'longer' and 'largest'; and MEASWORD for dimension adjectives like 'wide' and 'high'. There should also be a category for adjectives taking two or more arguments like 'disjoint' and 'equal'. Each of these has its own semantic function associating it with the noun it modifies. The ADJ's are sets which are intersected with the sets represented by the nouns. The function for ORDADJ chooses the appropriate element from a set based on its ordering. The function for COMPADJ chooses the largest or smallest from a set of alternatives. There is another function for the combination of an ORDADJ and a COMPADJ as for example 'the second largest factor of 12'. One important category of adjectives has not occurred in our subject matter. These adjectives express relative judgments about the noun that they modify. Two examples given by Sandewall [19] are:

"the little elephant"	(it is little <u>for an elephant</u>, but it may be big for an animal)
"the bad teacher"	(he is bad <u>as a teacher</u>, but he may be good as a father).

A discussion of a formal theory for these adjectives can be found in Montague [11].

The categorization of verbs is crucial in conversational speech, but has not been carefully worked on in our system due to the lack of variety of verbs in the present vocabulary.

One other important category in our system is ARITHREL for the arithmetic relations. These include 'less than', 'less than or equal to' (and similarly for 'greater'), 'equal to', 'equivalent to', and 'divisible by'. The variations like 'smaller than' and 'littler than' and the various abbreviations are handled by the TRANSL file. The prepositions are also eliminated by the TRANSL. The ARITHRELs are treated as functions. Thus 'less than 6' will be parsed by the rule

```
ARITHRELS <- ARITHREL NP        (APP ;1; ;2;) .
```

The result of applying the LESSTHAN function to 6 is the set of all real numbers less than 6 (represented of course by a characteristic function since it is an infinite set). These phrases can appear in several positions. Examples are:

EX1: Which factors of 12 are less than 6?
EX2: List all the factors of 12 that are less than 6?
EX3: Is 2 < 6?

The primary semantic function for EX1 and EX2 will be intersection and for EX3 will be subset. Basically these phrases which evaluate to sets can be used in the same ways as NP's which are sets, ADJ's which are sets, and the prepositional phrases like those with 'between' and 'before' which evaluate to sets. This greatly simplifies the grammar. All of these various phrases which evaluate to sets are included in the grammatical category SUBST:

```
SUBST <- NP
SUBST <- ADJ
SUBST <- ARITHRELS
SUBST <- PREPP  .
```

Thus the three general rules used in the derivations of the examples given above

```
Q <- INTER NP LINK SUBST
RELPRONS <- RELPRON LINK SUBST
Q <- LINK NP SUBST
```

will parse a large variety of sentences. It is also possible by using rules for lists of SUBST's to parse lists containing elements from the different types of phrases, for example:

Which factor of 12 is LEQ 6, divisible by 3, and even?

In addition this structure provides a convenient way to handle negation. The rule for 'not' is

SUBST _ /NOT/ SUBST (C ;2;) .

C is the semantic function for set complement. Examples of this are:

Give all the factors of 12 which are not divisible by 3!
Which prime number is not odd?

Careful attention to the semantics of a given grammatical construction leads to clarity in the grammar. The grammatical clarity is aided by both the lexical categories like N, FCN, ARITHREL, etc.. and the grammatical categories like SUBST which will be discussed in detail in later sections. In both cases the motivation for creating the categories is semantical as well as grammatical due to the close relationship of the syntax and the semantics.

Our goal has been clarity not only within the components of the system but for the system as a whole. The clear separation between the components of the system has not only added to its clarity but also furthered our other major goals of flexibility and extendability.

Flexibility is desirable in a system not only during the developmental stages but also as a property of the 'finished' product because of course a system is never really finished; one of any system's most important features is its extendability. The more flexible a system is the more easily it can be extended.

There are two basic types of additions that might be made to our question-answerer. The first are additions to the vocabulary and subject matter ranging from the simple addition of synonyms which can be handled entirely at the level of the TRANSL through the addition of new function words which need to be put in the dictionary and the function stored in the data base to the addition of completely new types of subject matter which would require new vocabulary, grammatical rules, semantic functions, and evaluation procedures. These sorts of additions should present no difficulties. The second basic type of extension would include substantive changes in the power of the system itself. These changes might be in areas like habitability, learning, modeling of the world and/or context of the conversation, and anaphoric references.

In discussing the question of clarity, examples were used from the system as a whole and from the syntactic and semantic components but not from the evaluation component. The evaluator is written in LISP, and as might be suspected, clarity is not its strong point. However this is the only area of our program which might in any sense be thought of as a 'black box' and it is certainly no less clear than the comparable components of other systems. In fact the semantic construction for a sentence is available as a sort of summary or outline of what the evaluator will do; so in that sense our program does provide a clear concise way to understand the general workings of the evaluator. However, the strong points of the evaluator are its flexibility and its independence from the natural language processing components.

One advantage of the independence of the components is that they can be programmed in different languages according to the appropriateness of the programming language to the task. The interfaces, the scanner and the parser are written in SAIL, which is fast, efficient and less space consuming The evaluator which deals with recursive functions many of which are created at runtime is written in LISP. The system is run on the PDP10 with TENEX and the fork structure of TENEX facilitates the running of separate components possibly in different languages. The problem of needing different programming languages for different tasks in the system is discussed by Bobrow in [2].

There are other advantages to the independence of the evaluation component from the natural language processing component. One area in which interesting work is being done is the area of representation of knowledge. As breakthroughs are made in this area, it will be possible for question-answerers to deal with much broader subject areas and in a much more efficient way. However, if the method used in the language processing component is based on the properties of the data base, then in order to take advantage of new ways of representing knowledge, the whole system must be rewritten. Woods discusses the importance of the independence of these components in [27].

> It seems that for efficient processing, different sorts of data require different sorts of data structures. A promising method for achieving reasonable efficiency in large less restricted universes of discourse is to provide the system with a variety of different types of data structures and special purpose deduction routines for different subdomains of the universe of discourse. Integrating a variety of special purpose routines into a single system however,

> requires a uniform syntactic and semantic framework. In general it is only after parsing and semantic interpretation have been carried out that such a system would be able to tell whether a sentence pertained to a given subdomain or not. Therefore, if the syntactic and semantic analyses were different for each subdomain, then the system would have to parse and interpret each sentence several times by different procedures in order to determine the appropriate subdomain. Moreover, there can be sentences that simultaneously deal with two or more subdomains, requiring a semantic framework in which phrases dealing with different subdomains can be combined.

Another important point raised here is that "different sorts of data require different sorts of data structures". This has not been implemented in most systems. For example, Quillian in discussing his Teachable Language Comprehender (TLC) says in [15]:

> TLC's second important assumption is that all a comprehender's knowledge of the world is stored in the same kind of memory, that is, that all the various pieces of information in this memory are encoded in a homogeneous, well-defined format. TLC's memory notation and organization constitute an attempt to develop such a format, which is uniform enough to be manageable by definable procedures, yet rich enough to allow representation of anything that can be stated in natural language. . . . It amounts essentially to a highly interconnected network of nodes and relations between nodes.

Historically, systems dealing with quite restricted subject matters and using evaluation techniques peculiar to the subject matter have been reasonably successful while systems attempting to cover diverse subjects using a generalized technique which will encompass different types of knowledge have been less successful. Woods' suggestion is that a general system instead of trying to find one method which will work for everything should recognize the differences in types of data and incorporate a number of different evaluation procedures in one program.

Our evaluator has the facility to do this. The data in our system can be stored in many different ways. Most of the data currently used are elementary math functions and therefore are well-suited to being stored as LISP functions which can be called when needed. Data can also be stored in tables and procedures involving table lookup implemented for determining measurement equivalences, etc. As the subject matter expands a variety of other data structures are possible. For example, Lindsay's SAD SAM program [9] was very successful at dealing with family relationships stored in a tree structure. Therefore, if we added family relationships to our program we might store the data on families in trees and program functions like 'father of' as tree search procedures. Another common way of storing data is on property lists; this could also be implemented. It is however interesting to note that portions of the use of property lists in early systems are now obsolete due to the newer technique of storing data as functions or procedures. For example, Raphael [16] gives as an example for SIR the description list for the number '3':

successor	4
odd	yes
shape	curvy

.
.
.

It is a waste of space to store the successor and the presence or absence of the property odd for every number. It is only necessary to store the successor function and the odd function and call them when needed. In fact it will probably someday be feasible to call a pattern recognition routine to determine the shape of a written number if that is desired.

One method often used by general systems is a theorem prover. Black [1] points out that theorem provers are designed for proving theorems not answering questions. He and others have developed deductive systems which are more compatible with the goal of answering questions. However, it is still true that most simple questions can easily be answered without any of the power of a theorem prover. In fact, the powerful machinery may often get in the way. On the other hand, certain questions are very well suited to handling by a theorem prover. For this reason, we plan to add a theorem prover to our system which can be called by the evaluator only when it is needed.

In the learning of elementary mathematics the student needs to acquire a large amount of methodological knowledge therefore it is reasonable to expect that a large portion of his or her questions would be of a methodological nature, for example, 'How do you find the factors of a number?' We have attempted in writing the evaluation procedures for functions to program them the way a student would do them rather than according to standard computer algorithms. We believe that the evaluator can therefore be made to answer methodological questions by analyzing its own functions. This is similar to Winograd's proposal [26] that his blocks program, in answer to a question about how it does something, be able to look at its own programs and convert them to an English description like "First I find a space, then . . . ."

II.2 Transformations

In the previous section, I discussed a variety of capabilities that our system has and some of the extensions that might be made. In this section, I will discuss the transformational abilities of our system. It is generally believed that a system based on an unaugmented context-free grammar cannot handle transformations. First I will explain why we do have the ability to deal with transformational constructions and then I will show how each of the constructions which cfg without semantic functions cannot handle are dealt with in our system.

It is of course true that a context-free grammar does not have the power of a transformational grammar, but the addition of the semantic functions allows one to 'recognize' semantically a non-context free set. Thus we have "augmented" our grammar by the addition of the semantic functions rather than by the addition of further grammatical apparatus. An example will make this point clearer. Consider the grammar G whose productions are as follows, where S, A, AND B are nonterminal symbols:

S <- A B
B <- z
B <- B z
A <- x y
A <- x A y

where $L(G) = \{x^n y^n z^m \mid m,n > 0\}$.

This is a context-free language. However by adding semantics to the grammar it is possible to tell for every string s, which is a member of L(G), whether or not s is in the set

$$\{x^n y^n z^n \mid n > 0\}$$

which is a context-sensitive set. Thus the combination of the grammatical rules with the semantic functions will recognize this context sensitive set. The semantic functions are as follows:

S <- A B	T if v(A) = v(B), NIL otherwise
B <- z	1
B <- B z	1 + v(B)
A <- x y	1
A <- x A y	1 + v(A)

hence, for s in L(G),

$$v(s) = T \text{ iff } s \text{ is in } \{x^n y^n z^n \mid n > 0\}$$

This example deals with context sensitive sets but the addition of semantic functions also can give transformational abilities. Specific examples will be given later in this section.

There are several advantages to using a context-free grammar. First, the format of the grammar is clear, easy to read, and universally recognized. Second, extremely efficient parsing algorithms exist for cfg. And finally, transformational grammars traditionally define a level of analysis called the syntactic deep structure which is described as follows by Fillmore [6]:

> It is an artificial intermediate level between the empirically discoverable 'semantic deep structure' and the observationally accessible surface structure, a level the properties of which have more to do with the methodological commitments of grammarians than with the nature of human languages.

Our system goes directly from the syntactic to the semantic representation, thereby eliminating the level of syntactic deep structure.

However, there are transformations which need to be made in the process. Woods [28] sums up the types of transformations needed as "reordering, restructuring, and copying of constituents". Restructuring will be discussed in detail in the next section with the examples being drawn from noun phrases with the verb 'have'. A simple example of reordering is the following:

```
C <- /DIVIDE/ NP /BY/ NP        (DIV ;2; ;4;)
C <- /DIVIDE/ NP /INTO/ NP      (DIV ;4; ;2;)
```

Thus 'Divide 6 by 3!' and 'Divide 3 into 6!' will both have the semantic construction (DIV (LST 6) (LST 3)) reflecting the fact that they have the same meaning.

Copying can also be done quite simply, although it is not often used in our system because of semantic functions like CHL which will be explained shortly. An example of copying is the following rule:

RULE1: NP <- DET ADP /OR/ DET ADP NP
(CHL (I ;2; ;6;) (I ;5; ;6;))

Ex. Is 2 an even or an odd number?

The first occurrence of the word 'number' has been dropped by the questioner but must be put back before it can be evaluated. The word 'number' which is the sixth element in the NP therefore appears twice in the semantic function in the proper positions.

CHL (choicelist) is used for many of the 'or' constructions (see Chapter IV for a discussion of lists with 'and' and 'or'.) When the argument to one of the semantic functions begins with CHL, the function is performed on each of the arguments in the list and an answer returned for each. In the above example the semantic construction will be

(QUS (S (LST 2) (CHL (I @EVEN @NUMBER) (I @ODD @NUMBER)))).

{Note: S is the subset function.}

{2} will be checked to see if it is a subset of the set of even numbers and then to see if it is a subset of the set of odd numbers; and the answer will be

(CHL (TV T) (TV NIL)).

The CHL is pushed outward as the evaluation proceeds. The reason for this will be clear in the next example.

As I mentioned above, our system does not often use copying because CHL can be used instead. The phrase 'an even or an odd number' could be given the semantic function (I (CHL ;2; ;5;) ;6;). The result of evaluating this portion, since the CHL is moved outward after the two intersections are performed, will be the same as the result after evaluating the two intersections in the semantic function for RULE1 above.

The CHL function could probably be viewed as either copying or restructuring. Before I discuss more examples of restructuring in the next section, I should mention briefly several other common criticisms of cfg.

Woods [28] points out that

> The unaided context-free grammar model is unable to show the systematic relationship that exists between a declarative and its corresponding question form, between an active sentence and its passive, etc. Chomsky's theory of transformational grammar, with its distinction between the surface structure of a sentence and its deep structure, answers these objections but falls victim to inadequacies of its own . . . .

It should be clear from the examples used throughout this paper that the semantic constructions for related inputs will show the deep structure relationship.

Work has been done by Postal [13] and others to isolate constructions found in natural languages which can be proven to be beyond the capacity of cfg. Postal has done this for constructions which have what he calls the property [xx]. His work was done with the Mohawk language but he also cites as an example in English the construction with 'respectively'. This construction is also mentioned by Winograd [26] as one that is impossible for a cfg. His example is

John, Sidney, and Chan ordered an eggroll, a ham sandwich,
and a bagel respectively

We have not included this construction in our grammar but it could be handled by a semantic function which counted the elements on both lists to check that the input was correct and then formed the proper ordered pairs with one element from each list.

Another serious objection to cfg is stated by Chomsky [3]:

> Immediate constituent analysis has been sharply and, I think, correctly criticized as, in general, imposing too much structure on sentences.

The most frequently cited examples as evidence for this claim are constructions with an unbounded number of immediate constituents. It is impossible for a cfg to handle these constructions correctly without an infinite number of rules of the form:

```
X <- A
X <- A A
X <- A A A  .
```

In order to handle these constructions in a finite grammar, rules are written of the form:

```
X <- A
X <- A X
```

which put the elements of the input at different levels of the parse tree rather than all on the same level, thereby imposing a structure which was not in the original input. The common example of this for natural languages is a string of adjectives. Here, again, the problem is solved by the semantic functions. This sort of string of adjectives will be a string of adjectives representing sets and they should be intersected, for example, 'the even prime numbers'. The intersection function will appear only once with the arguments passed up to it as they are parsed.

```
ADP <- ADJ                          ;1;
ADP <- STRINGADJ ADJ                (IER ;1; ;2;)
STRINGADJ <- ADJ                    ;1;
STRINGADJ <- STRINGADJ ADJ          ;1; ;2;
```

[Note: IER is a special intersection function which uses heuristics to perform the intersections.]

II.3 Restructuring

The most complex semantic functions are those associated with restructuring. Woods [28] describes the contrast between the ordinary control structure associated with cfg and the structure in his system.

> In ordinary context-free recognition, the structural descriptions are more or less direct representations of the flow of control of the parse as it analyzes the sentence. The structural descriptions assigned by the structure building rules of an augmented transition network . . . are comparatively independent of the flow of control of the algorithm. This is not to say that they are not determined by the flow of control of the parse, for this they surely are; rather we mean to point out that they are not isomorphic to the flow of control as in the usual context-free recognition algorithms.

Our approach differs from Woods' but the purpose is the same. The order of the content words will be roughly the same in the semantic function as in the original input reflecting the order in which they were parsed, but for those sentences requiring restructuring the primitive semantic functions used will provide the evaluator with a control structure for dealing with the arguments which reflects the deep structure semantics rather than the surface parse. The need for this sort of semantic function was first discovered when dealing with the following examples:

EX1: Which is a factor of 4: 2 or 8? answer: 2
EX2: Which has a factor of 4: 2 or 8? answer: 8

The words are identical except for the verb yet the evaluation procedures are quite different. In EX1 FACTOR is applied to 4 while in EX2 it should be applied instead to both 2 and 8. The problem is to block the application of FACTOR to 4 in EX2. Our initial idea was to add a semantic function when the verb was parsed which would alert the evaluator to change the order of the arguments for the remaining part of the semantic function. There are at least three major drawbacks to this approach. First it has no flexibility and would require a large amount of coding for all the possible noun phrases and hence semantic functions that might fill the remaining slots in the sentence. Second, it would destroy the straightforward character of the semantic construction since portions of the construction would not be what they appeared to be. Third and most important it would destroy the recursive inside-out nature of the evaluator. The alternative that we have chosen is to use separate noun phrase rules and hence separate semantic functions to parse the noun phrase 'factor of 4' in the two examples. This adds to the size of the grammar but does reflect the difference in meaning of the noun phrase in the two examples. We therefore have two sorts of NP rules in our grammar: the regular NP's and a type that we call HNP's. We expect that the use of the HNP's will be wider than merely in connection with the verb 'have' (for example with relative possessive pronouns). The rule that parses EX2 is

```
Q <- INTER /HAVE/ HAVENP PUNCHOICE CHOICELIST
```

[Note: PUNCHOICE allows a variety of punctuation marks.]

There are several types of HNP's, one of which is HAVENP. The semantic functions for some of them create sets and for others functions. In this case a set will be created which contains all the numbers that have 4 as a factor, then the singleton sets containing 2 and 8 respectively can be checked to see if they are subsets of this set.

This approach appears to be compatible with a comment made about 'have' by Winograd [26]:

> The interesting thing about "have" is that it is not used to indicate a few different relationships, but is a place-marker used to create relationships dependent on the semantic types of the objects involved.

The simplest kind of HNP is called FCNHNP:

```
(44,5)  Q1 <- INTER FCNHNP AUXIL NP /HAVE/
                           (APP ;2; ;4;)
```

EX3: What factors does 12 have?
EX4: Which even factors that are less than 6 does 12 have?

The rules that parse FCNHNP's are:

```
(123,1) FCNHNP <- FCN                   ;1;
(123,2) FCNHNP <- ADP FCN               (FCNMK ;2; ;1;)
(123,3) FCNHNP <- FCN RESTRICT          (FCNMK ;1; ;2;)
(123,4) FCNHNP <- ADP FCN RESTRICT  (FCNMK ;2; (I ;1; ;3;)) .
```

In the simplest case like EX3 where there is only the name of a function, nothing unusual needs to be done. Rule (44,5) applies the FACTOR function to its argument which is 12. EX4 is more complicated. Through the semantic function FCNMK, a new function is created at runtime which can be applied to 12 by Rule (44,5). FCNMK takes two arguments. First the name of the function, i.e., FACTOR and then a set representing the various restrictions given by the adjective modifiers and the other restrictive clauses (RESTRICT) which can be relative clauses, prepositional phrases or arithmetic relations. Each of these restrictions is a set so they can all be intersected into one set to fill the second argument slot for FCNMK. To standardize the notation in this and the following examples, I will give the format for FCNMK as

(FCNMK f s)

meaning it has two arguments the first f a function and the second s a set. From these two arguments it creates a new function. In the case of EX4 it will create the function which when applied to a number returns all its factors that are both even and less than [illegible].

There are four types of HAVENP's which create sets. The first type contains an existential quantifier either implicitly or explicitly. Examples are:

EX5: Does 6 have any odd factors that are not divisible by 3?
EX6: Does 5 have an even factor?

The examples studied indicate that the indefinite article in these contexts should be treated as an existential quantifier. The semantic function with its arguments is

(EXTHNP f s) .

The first argument is a function which will be parsed by FCNHNP, discussed above, and therefore is either a function in the data base or a new function to be created at runtime by FCNMK which incorporates the restrictions on the function, for example, in EX6 the function will be the EVENFACTOR function. The second argument is a set and in this particular case it will always be by default the universal set. The new set created by EXTHNP is

{x | f(x) INTERSECTION s is nonempty}

which is logically equivalent to

{x | (EXISTS y in f(x)) (y in s)} .

Note that intersecting a given set with the universal set has no effect. The second argument to EXTHNP is used in cases where there is an exception given, for example

EX7: Does 12 have any odd factors other than 3?

Here the first argument to EXTHNP is again the function, perhaps one created by FCNMK, and the second argument is a set which is the complement of the set given in the exception clause. We do not consider 1 to be a factor of any number since factor was the first function we programmed and there are not as many interesting questions that can be asked if 1 is considered a factor of every number. Therefore in EX7, f is the ODDFACTOR function and thus f(12)={3} which intersected with the complement of {3} equals the empty set. Therefore 12 does not belong to the set created by EXTHNP.

The universal quantifiers used are 'all' and 'only', as in

EX8: Does 9 have all odd factors?
EX9: Does 12 have only even factors except for 3?

The function UNVHNP is more complicated than EXTHNP because FCNMK cannot be used. FCNMK makes new functions by incorporating the adjective and other restrictions into the function. With the universal quantifier this cannot be done. For example, in EX8 we do not want to know if 9 has any odd factors. Instead we want to first find the factors of 9 and then make sure that all of them are odd. This adds to the grammar because all of the rules given for FCNHNP above which parse the various kinds of modifiers must be duplicated here. The format for UNVHNP is

(UNVHNP f Sr)

where f is a function in the data base and Sr is the set of restrictions. UNVHNP creates the set

$\{x \mid f(x) \text{ is a subset of } Sr\}$.

A separate semantic function is used for universal quantifiers where an exception set is given as in EX9.

(UNVHNPXCT f Sr Sx) .

It has 3 arguments, a function in the data base, the set of restrictions, and the exception set. It creates:

$\{x \mid (f(x) - Sx) \text{ is a subset of } Sr\}$

In EX9, $f(12)=\{2,3,4,6,12\}$ and the set difference of this with $\{3\}$ is $\{2,4,6,12\}$, which is a subset of the set of even numbers.

The third type of HAVENP's use a numerical determiner. For example

EX10: Does 12 have 2 odd factors?

uses the semantic function

(EXPHNP <number> f)

which creates the set

$\{x \mid \text{CARDINALITY}(f(x)) = \text{<number>}\}$

Here f may again be a function created by FCNMK.

The final type of HAVENP are called ANSHNP because the desired result of the application of the function is included in the HNP, for example

EX11: Does a 2 inch square have an area of 4 square inches?
EX12: Does 4 have 2 as a factor?
EX13: Does 1/2 have the denominator 2?

EXTHNP which was discussed above in the section on existential quantifiers is used also for the ANSHNP. The function f is handled by FCNHNP using FCNMK and the set s is the desired result given. The set created is again

$\{x \mid f(x) \text{ INTERSECTION } s \text{ is not empty}\}$

In EX12, $f(4)=\{2\}$ and $\{2\}$ INTERSECTION $\{2\}$ is not empty, so the answer to the question is TRUE.

Chapter III

CONSTRUCT and the Grammar

III.1 CONSTRUCT

CONSTRUCT is the master program for the question-answering system. It is written in SAIL and provides the interface between the natural language processing component and the evaluator. It also contains the SCANNER and handles the actual parsing. The grammar is read in as a file and compiled before it is run for greater efficiency. When the parse is finished, the semantic construction is in abbreviated notation. It is then prepared for the evaluator by a macro expander.

In addition to its function of running the question-answerer, CONSTRUCT also provides interaction with the user at runtime. The user has an option of several modes of operation. The basic choice is between file or teletype input and output. It is also possible to run without either the TRANSL, the dictionary or the evaluator. This might be useful for testing whether a list of lexical forms will parse, however, basically it is designed for the work that CONSTRUCT does with other forms of grammatical analysis rather than for the question-answerer. CONSTRUCT is a versatile program which is part of a package of programs for natural language processing, written by R. Smith, that have served a number of users for a variety of purposes.

Another useful option is the output of a cleaned-up version of the grammar file. With several people working on the project, the grammar file often becomes overloaded with comments and notes and the rules out of any intelligible order as additions and deletions are made. CONSTRUCT will print out the grammar without comments and with the rules grouped according to the left-hand-side symbol. It also automatically provides a numbered label for each rule.

While the program is running, CONSTRUCT provides for editing of the typed input in case of typographical errors. The experienced user can also go into DDT and change storage and other parameters and then return. The TRANSL file can be replaced with a different version. A word can be added to or deleted from the dictionary or given a new category; and if the change is to be permanent, the program can be requested to write the new version of the dictionary on the file storage device so that it will not be lost when the program is exited from.

Other features of the program are designed to aid in debugging the grammar. The start symbol can be changed so that various phrases can be tested. A printout of a group of rules can be requested. The printout of a particular derivation can be aborted and the next derivation begun or control can be returned immediately to the user. The previous input sentence can be redone by a single character command rather than retyping the sentence. This is useful on a display terminal where there is no paper printout to refer back to.

Often while running the question-answerer, interesting features are discovered about the way the system handles certain questions. If a particular derivation is noteworthy for some reason, CONSTRUCT can be requested to send the question to a file of the user's choice. In this way, separate files can be maintained for questions which are good examples of the question-answering system at work, questions which parse correctly but are evaluated incorrectly, and questions which fail to even parse. The latter files can be used for diagnostic purposes and the former to demonstrate the system to visitors.

III.2 The Scanner and the Dictionary

The scanner used by CONSTRUCT is similar to the scanners found in compilers. It preprocesses the input before passing it to the parser in the form of a string of lexical categories. The punctuation and arithmetic signs in the input are passed on untouched. A table of break characters is used to identify the word boundaries. Numbers are assigned their lexical category, either INTEGER or REAL, directly by the scanner. The lexical categories for other words are looked up in the dictionary. If a word has multiple categories in the dictionary, all the alternatives are entered in the lexical representation for the string. For example

2 is a prime.

will be represented as

INTEGER LINK /A/&VAR ADJ&N .

[Note: VAR is the category for variables and /A/ for the indefinite article.]

There are currently over 300 words in the dictionary and only 30 have multiple categories. Of these, 11 have been satisfactorily worked out in the grammar and cause no real problems. They include:

1) 5 ADJ&N like 'real' and 'prime';

2) 3 variables:

a) a --- VAR and indefinite article;

b) x --- VAR and /BY/ as in '2x4 inch rectangle';

c) n --- VAR and notation for cardinality as in 'n {a,b,c}=3';

3) 'Which' as both an interrogative and a relative pronoun;

4) 'Square' as GEOFIGURE as in 'a 2 inch square' and as /OPER/ as in '2 square feet';

5) 'May' as either the name of the month or a modal verb (note: some terminals cannot distinguish between upper and lower case) .

The other 19 are in areas of the grammar that are unfinished. Six of the words with multiple categories are verbs. Our vocabulary included too few verbs to do extensive categorization on the basis of the underlying semantics. These categories are therefore somewhat makeshift, but cause no problems. Two of the words are 'intersection' and 'union' which have separate categories for the two possibilities: 1) the function name with a list of arguments as in 'the union of {a} and {b}' or 2) the function name in infix notation as in '{a} UNION {b}'. Three of the words (and there are probably many more) are 'day', 'week', and 'month'. These examples were discussed in Section II.1. They can apparently be either N's or FCN's, but more work may show that creating a new single category would be a better approach. No rules have been developed in the grammar yet for them. The remaining eight multiple category words are all some variation of written out numbers. We have chosen eight of the more common ones to put in the dictionary for the purpose of testing the grammar. Algorithms exist for easy conversion of written numbers and this conversion should properly be performed as part of the scanner. The difficulty lies in the ambiguity of certain forms. For example, 'one' can be used in two ways:

EX1: There is only one even prime number.
EX2: 231 = 2 hundreds 3 tens and 1 one.

Similarly 'fourth' has two uses:

EX3: The fourth largest factor of 12 is 3.
EX4: 1 fourth = 2 eighths.

It is interesting to note that in both EX2 and EX4 the problem only arises in the singular and for the ORDADJ's it only arises beginning with the third, i.e., first, second, third vs. ..., half, third. Informal questioning of foreign speakers indicates that this is not a problem in every language.

The vast majority of words in the dictionary, however, have only a single lexical category. The multiple categories caused a large amount of grammatical ambiguity with early versions of the grammar. As the precision of the grammar has increased these have virtually disappeared. The only remaining ambiguity is for the multiple category ADJ&N, for example,

EX5: Is 2 prime?
EX6: Is 2 a prime?
EX7: Are 2 and 3 prime?
EX8: Are 2 and 3 primes?

EX6 with the determiner will have only a single parse, but the other three examples will parse 'prime' as both a noun and an adjective. There is no ambiguity for the native speaker because of the verb tense and the determiner used with the singular to make it agree with the verb and the 's' ending used with the plural for agreement. Our system uses only the singular form of nouns and verbs (the standardization is done by the TRANSL) and hence has no facility for checking agreement. The mathematical subject matter has no semantic need for tenses. The grammar would be more precise if they were included, but the processing time would be increased out of proportion to the advantages gained. Note that the semantic construction will be the same for both parses in each of the above examples, therefore, the ambiguity is no problem. The use of tenses and agreement in a grammar supplies it with the power to convey certain features of meaning. These features of meaning are not present in elementary mathematics which might be called 'tenseless'. The features remain in the grammar but their potential power is not actualized for the semantics, therefore, we have chosen to ignore them thereby losing some of the ability to discriminate between grammatical forms that have the same meaning. Note that this also means that the input grammar cannot distinguish between correct syntax and certain forms of incorrect syntax. For this reason, tenses and agreement will need to be included in the output grammar so that it will produce only grammatically correct sentences.

III.3 The TRANSL File

The scanner checks to see if any word or group of words in the input is in the TRANSL file before it looks up the lexical categories of the words in the dictionary. The TRANSL contains strings which are to be substituted for and the substitution which may be the empty string. There are five basic uses of the TRANSL in the current version of the system. 1) All plural forms are TRANSL'd to the singular. 2) All abbreviations are TRANSL'd to the full word singular form. 3) Synonyms are TRANSL'd to the most commonly used one of the group. 4) Two or more words which always occur together, one or more of which may have no meaning alone in the particular subject matter, are TRANSL'd to a single word representation, for example, 'wholenumber'. And 5) noise words are eliminated. Some of the noise words are in the nature of interjections which have little meaning in any subject matter. Others are words like 'also' and 'both' which in ordinary conversation add precision and shades of meaning but are unneeded in mathematics which already uses a precise rigid approach to the determination of meaning. It is for this reason that so few adverbs are used at all in mathematical language.

EX1: Find the even number which is a factor of 4 and 6.
EX2: Find the even number which is a factor of both 4 and 6.
EX3: Find the even number which is a factor of 4 and also a factor of 6.

The use of 'both' and 'also' in these examples adds nothing to the meaning so they are eliminated at the level of the TRANSL.

Some analogies can be made between the function of our TRANSL file and certain features of other natural language processing systems, in particular, pattern recognition systems. Bobrow [2] distinguishes between 'structural' transformations and 'definitional' transformations. He gives as examples of definitional transformations, the substitution of 'twice' for '2 times' and 'one half of' for '.5 times'. In our system, the TRANSL handles these sorts of 'definitional' transformations. The TRANSL also performs another function in the same way as a system based on the pattern recognition technique. Colby and Enea [4] give the example:

Could you tell me your name?

The literal analysis is obviously incorrect. Rather, in polite conversation, certain standard phrases which add no meaning are used to introduce questions. Colby and Enea use the following rules to deal with these phrases:

```
RULES OF SENTENCE =
   <QUESTION-INTRODUCER>: Q <NOUN-PHRASE>:N
                          ← (IS :N '*'?'* );
RULES OF QUESTION-INTRODUCER =
   COULD YOU TELL ME ←,
   WOULD YOU TELL ME ←,
   PLEASE TELL ME ←,
```

Our TRANSL includes the two strings 'do you know' and 'can you' which are very similar to these examples. It is interesting to note that their system which is designed for the pattern recognition technique requires as many rules to deal with the question-introducer as ours does. We need one line in the TRANSL for each of the phrases as they need one rule for each; and we also need one grammatical rule for each sufficiently different construction of the question following the phrase. We have the two rules:

RULE1 Q ← /DOYOUKNOW/ NP
EX1: Do you know the sum of 2 and 4?

RULE2 Q ← /DOYOUKNOW/ INTER NP LINK
EX2: Do you know what the factors of 12 are?

RULE1 is analogous to their rule and they would need to add RULE2 before they could handle EX2.

III.4 The Grammar

The grammar is a context-free grammar. Winograd [26] in discussing augmented transition networks says

> The advantages lie in the ways in which these augmented networks are close to the actual operations of language, and give a natural and understandable representation for grammars.

This is also the goal of our grammar and associated semantic functions. By writing the rules so that appropriate semantic functions can be assigned to them, the rules themselves are more natural and closer to the "actual operations of language. The trees are considerably flatter than the trees for parses by more conventional context-free grammars used for natural language processing. The start symbol S parses an input sentence according to its type: Q for questions, D for declaratives, C for commands, and F for arithmetic formulas. At the top level for each of these categories, the sentence will be parsed by a rule that shows its basic structure in considerably more detail than the usual S ← NP VP. Because the grammar needs to 1) determine the correct semantic function and 2) locate all its arguments, there will be no categories in the grammar like VP which are complete 'black boxes'.

Montague [11] argues against the approach of attacking syntax first and then considering semantics.

> Such a program has almost no prospect of success. There will often be many ways of syntactically generating a given set of sentences, but only a few of them will have semantic relevance; and these will sometimes be less simple, and hence less superficially appealing, than certain of the semantically uninteresting modes of generation. Thus the construction of syntax and semantics must proceed hand in hand.

A word of caution is also needed for those who would attack semantics first. The guidelines for semantic interpretation are established by the syntax. If an attempt is made to analyze meaning in isolation from the syntax, there is also almost no prospect for success. It is possible to write a jumbled program that will handle bits and pieces of the input that it picks out, and even do fairly well on the limited set of sentence types for which it was designed, but no organized, flexible, general semantic approach can be constructed which is not closely guided by the syntax.

Montague says that the rules with semantic relevance may be "less superficially appealing". Our experience has shown that the appeal of semantically poor rules is very superficial indeed. For small portions of the grammar, more efficient and appealing rules can certainly be written, however, the only way to keep the various areas of the grammar from causing grammatical ambiguities and other difficulties when they work together is by considering the semantics at every step. This is the structure that grammar has. If an attempt is made to parse natural language with a grammar that is appealing on some other ground, it simply cannot be made to fit the language.

Therefore, our primary consideration in writing the rules of the grammar is to facilitate the writing of semantic functions for the rules. The next consideration is to write rules which minimize grammatical ambiguity. Given that these two conditions are satisfied, other factors such as the number of rules required to parse a particular construction can be considered.

At this point, an illustrative example of this grammar writing procedure will be helpful. Consider the following questions:

A: Is 2 odd?
Is 2 an even number?
Is 2 greater than 1?
Is 2 between 1 and 12?

B1: Which factors of 12 are odd?
Which factors of 12 are even numbers?
Which factors of 12 are greater than 1?
Which factors of 12 are between 1 and 12?

B2: Which factors of 12 are not odd?
Which factors of 12 are not even numbers?
Which factors of 12 are not between 1 and 12?
Which factors of 12 are not between 1 and 12?

First we can notice that the adjective 'odd', the noun phrase 'an even number', the arithmetic relation 'greater than 1', and the prepositional phrase 'between 1 and 12' each have the same semantic role in the A-group questions and also have the same role as each other in the B-groups. The primary semantic function for the A-group is subset and the primary semantic function for the B1-group is intersection. So we can write the following rules for use in these two contexts as well as many other contexts:

```
SUBST <- NP
SUBST <- ADJ
SUBST <- ARITHRELS
SUBST <- PREPHRASE
```

[Note: Only a few of the prepositions fall under this particular category of prepositional phrases.]

Next, we can consider how to handle the occurrence of 'not' in the B2-group. There are four plausible ways of writing these rules if only the syntax is considered.

```
(1)     Q <- INTER NP LINK SUBST
        Q <- INTER NP LINK /NOT/ SUBST

(2)     Q <- INTER NP LINKNOT SUBST
        LINKNOT <- LINK
        LINKNOT <- LINK /NOT/

(3)     Q <- INTER NP LINK SUBST
        ADP <- /NOT/ ADP
        NP <- /NOT/ NP
        ARITHRELS <- /NOT/ ARITHRELS
        PREPHRASE <- /NOT/ PREPHRASE

(4)     Q <- INTER NP LINK SUBST
        SUBST <- /NOT/ SUBST
```

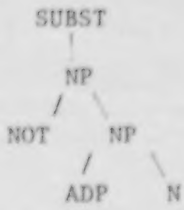
The last proposed set of rules is the one we have chosen. The second would be the worst choice because no semantics could be written for the rules. Failure to support a semantic analysis is the worst problem that a grammatical rule can have. The 'not' in the second alternative is simply buried too deeply in the grammar. It cannot be semantically attached to the SUBST or to the relationship between the NP and the SUBST.

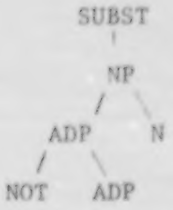
The first set of rules could form the basis for a workable semantics for 'not' although the grammar would need to be longer since each rule containing a SUBST would need to be duplicated with /NOT/ inserted before SUBST and a new semantic function written. For example here the semantics for the positive case would be intersection and the semantics for the negative case could be either intersection with the complement or set difference. Also a complete set of listing rules like

```
Q <- INTER NP LINK /NOT/ SUBST /AND/ /NOT/ SUBST
```

would be needed at each top-level position. Using the fourth alternative the listing (and choice) rules need to be given only once at the SUBST-level.

The third alternative has the same end result as the preferred fourth alternative, but it can be rejected on the grounds that it is longer. Also it would lead to syntactic ambiguity in the case of SUBST's which are noun phrases containing an adjective. For example 'not even numbers' could be parsed as either:





.

All of the groups of rules in the grammar will be given and briefly described in the next chapter. In the remainder of this section, I will discuss the general problem of ambiguity. There are a variety of types of ambiguity. There are also a variety of approaches available for disambiguation. No single approach can handle all types of ambiguity. Our intention is to disambiguate by combining several approaches in a natural way so that the disambiguation for any given sentence will correspond as closely as possible to the way a native speaker would perform the disambiguation. Because our grammar cannot hope to be as complex or the heuristics in the evaluator as broad in scope as a person's, certain of the disambiguation techniques which are of minor importance for a person will carry a disproportionate share of the workload in our system. Two of these techniques which are probably not as often used by native speakers may unfortunately play a large role in our system. The first is the case where a person is so unsure of the intended meaning that he or she must ask for clarification. The second case is where the person is fairly sure of the intended meaning but does recognize that there is another possible interpretation which has some plausibilty. In this case, the person may ask the speaker if the more probable interpretation is in fact the intended meaning or , especially in the case of written speech, may adopt the interpretation with the highest probability subject to revision if further input shows it to be the wrong assumption.

I will distinguish three types of ambiguity and call them lexical, grammatical, and semantic ambiguity. At the surface level before any syntactic or semantic analysis has been done, there may be a fair amount of lexical ambiguity, i.e., words with multiple lexical categories. The vast majority of these cases are disambiguated on the basis that only one will parse. If there is more than one parse, there will still be no problem, as in the case of ADJ&N, if both parses yield the same semantic construction. Even if there is no lexical ambiguity, an input may have more than one syntactic parse. If the derivations all produce the same semantic construction, I will call it grammatical ambiguity, and if different semantic constructions are produced, I will call it semantic ambiguity. It is also possible to have semantic ambiguity even if there is only one parse as in Katz' 'colorful ball' example. This ambiguity cannot be detected by the natural language processing component of the system. It will be detected and resolved by the evaluator on the basis of context and world knowledge. Therefore I will not discuss this form of ambiguity in this section. The ambiguity we are concerned with here is ambiguity in which the meanings of words are not in dispute only the way of combining them. As grammatical ambiguity is naturally resolved by the semantics in the case where only one semantic construction is produced, similarly, semantical ambiguity is naturally resolved by the evaluator when either the evaluation is the same or only one of the semantic constructions can be successfully evaluated. These sorts of disambiguation, the grammar settling lexical ambiguities, the semantics settling grammatical ambiguities, and the evaluator settling semantic ambiguities, are a natural by-product of the structure of the system. The ambiguities which remain require additional measures. Sometimes a sentence will have two or more parses with different semantic constructions and the output from the evaluator is also different for each. The addition of context checking and world knowledge to the evaluator will cut down the number of these remaining ambiguities by discarding more possible interpretations as unacceptable at the level of evaluation. Also certain heuristics can be added which are based on insights about the sorts of questions that people are likely or not likely to ask. Failing any of these measures: 1) the user can be asked what was meant, 2) the interpretation which has the grammatical structure with the highest probability based on a probabilistic fit of the grammar with a corpus of sample questions can be tentatively chosen, or 3) the user can be presented with the most probable interpretation selected in this way and asked if it is correct. Work by R. Smith [21] with probabilistic grammars shows that in many cases they can be used very successfully for disambiguation. Through the use of complicated analysis and statistical programs which will not be discussed here, probabilities can be assigned to each rule on the basis of its group membership determined by left-hand-side symbol (the probabilities of the rules in each group summing to 1) so that the probability assigned to each derivation when there are multiple derivations of a grammatical construction will accurately reflect the percentage of times that it was the correct derivation on semantical grounds in a large sample set of sentences containing that grammatical construction. Thus by analyzing large numbers of sample questions, we can know for a given ambiguous sentence type which grammatical parse is more frequently the correct one. The CONSTRUCT program is prepared to handle the addition of probabilities to the grammar. The use of probabilities has an analogue in human processing. A person will first understand a sentence according to commonly used grammatical constructions and then if there is a problem reevaluate it with more uncommon grammatical rules. I will conclude this section with an example of the handling of the ambiguity problem for lists of units.

The rules for lists of UNIT s need to be written carefully to avoid unnecessary grammatical ambiguities and detect genuine semantic ambiguities. Consider the following lists of units:

EX1: 5 yds 3 ft 2 in
EX2: 2 lb 3 oz, 4 lb, and 5 oz
EX3: 2 yds and 2 ft

Lists with no commas, such as EX1, are clearly intended to be compound units rather than a list of single units. These are parsed by a grammatical category called JOINUNITS with the semantic function

UNITJOIN. The evaluation procedure for UNITJOIN adds the units together and forms a single new unit for the evaluator to work with. EX2 is a list of three elements (one of which is a compound unit) and no further joining should be done among the three top-level elements. EX3 can in some contexts be genuinely ambiguous. There is no method in our system for attaching two possible semantic functions to one rule of the grammar. It is also not either possible or desirable for one semantic function to be evaluated in two completely different ways. So the solution chosen here was to write syntactic rules which would be known to generate two-way ambiguities in certain cases with each of the ambiguous derivations assigned a different semantic function. Consider the command

Convert 2 yds and 2 ft to inches.

The answer could be either

(CMD (CHL 72 INCHES 24 INCHES))

or

(CMD 96 INCHES) .

The first answer results from parsing by LISTOFEXP which uses the CHL semantic function and thus gives an answer for each of the elements on the list. The second answer results from parsing by SUMUNITS which like JOINUNITS uses the UNITJOIN semantic function and thus adds the units together before it performs the conversion. Sentences like these will be derived in both ways thus reflecting the genuine ambiguity. EX2 will be parsed only by LISTOFEXP because SUMUNITS does not allow a joined unit as an element since the presence of a compound unit on a list blocks further joining. However, sometimes there are other clues in the input which allow the native speaker to disambiguate it. Consider the following commands:

EX4: Convert 5 yds, 3 ft, and 2 in to inches!
EX5: Convert 5 yds, 3 ft, and 4 ft to inches!

These commands are easily understood by the native speaker. The derivation using SUMUNITS is the correct one for EX4 and the derivation using LISTOFEXP is correct for EX5. In this case, heuristics should be added so that the evaluator will reject the incorrect derivations and eliminate them at the evaluation level rather than trying to eliminate them at the syntactic level. For example, it would be an error when the evaluator was asked to convert 2 inches to inches in EX4, and an error when asked to join 3 feet and 4 feet into a compound unit in EX5.

There is one other possible source of unacceptable ambiguity. Consider:

EX6: A line 2 yds, 1 ft, and 10 in long
EX7: A triangle with sides 2 yds, 1 ft, and 10 in

The units in EX6 should be joined and those in EX7 should not. The evaluator will reject the incorrect parses on the basis of the number of arguments that it expects for working with lines and triangles. It might be possible to put these argument slots into the grammar explicitly, but it would lengthen the grammar too much to be practical.

Chapter IV

The Rules of the Grammar and their Semantic Functions

IV.1 Introduction

The grammar contains 642 rules. As I discuss the rules, I will note areas where additions are planned. Some groups of rules are known to be incorrect and I will point these out and indicate what remedies are needed. Other parts of the grammar do not yet have associated semantic functions so their correctness cannot be tested. A rule of the grammar can be incorrect in either of two ways. First it can fail to parse the constructions it was intended to parse or second it can parse in an unnatural way so that no semantic function can be written for the rule. It has generally been true that writing the semantics brings to light problems with the grammar -- some of them trivial problems but others have been major, unexpected, and in general very revealing about the nature of the language that we are dealing with. One striking case where this happened involved sentences containing the verb 'have'. [See Section II.3].

In the next sections I will go through the grammar one group of rules at a time and give a short description and explanation of the rules in the group, including examples and any information about their evolution through the various revisions of the grammar that might be of interest.

The rules are labelled by an ordered pair of numbers enclosed in parentheses. Each rule with the same left-hand-side (lhs) has the same first number in the label and the various rules in the group receive consecutive second numbers. This structure was created to handle probabilistic grammars where groups with the same lhs need to be distinguished. If and when we decide to implement probabilities the CONSTRUCT program is prepared to handle them.

The rules of the grammar are given in standard context-free rule format with the lhs and rhs separated by '<-'. Each rule is followed by a tab and then its associated semantic function. Numbers enclosed in semi-colons in the semantic function refer to the position of elements in the rhs of the syntactic rule. Numbers without semi-colons are passed directly to the evaluation program and are usually default values for argument slots. Perhaps the best way to show how the semantic functions work is by a simple example which uses only the following rules:

```
(1,2)   S <- F                         (FML ;1;)
(2,1)   F <- F2                        ;1;
(2,2)   F <- LISTOFF                   (ANDER (LST ;1;))
(3,4)   LISTOFF <- LISTOFF , /AND/ F2  ;1; ;4;
(3,2)   LISTOFF <- F2 , F2             ;1; ;3;
(4,2)   F2 <- EXPP ARITHREL EXPP       (S ;1; (APP (FCN ;2;) ;3;))
(7,1)   EXPP <- EXP1                   (LST ;1;)
(13,1)  EXP1 <- EXPT                   ;1;
(14,1)  EXPT <- EXPF                   ;1;
(15,2)  EXPF <- TERM                   ;1;
(16,1)  TERM <- INTEGER                ;1;
```

To show how the semantic construction which is to be passed to the evaluation program is built up, I will go step by step through the syntactic parse and show what information is added to the semantic construction at each step. I will use two examples:

```
EX1:  2<3
EX2:  2<3, 3<4, and 4<5  .
```

For this purpose I will abandon strict left to right processing and instead process the integers simultaneously by multiple use of the rules where possible. When an argument in the semantic construction is not yet specified I will represent it by [ ]. And when a terminal symbol in the grammar is reached and there is a corresponding [ ] in the semantic construction, I will put the symbol in the brackets, for example, [integer]. Before this is passed to the evaluation program, the actual integer in the input sentence will be substituted into the formula by a macro expander. Terminal syntactic categories like /AND/ which serve to give structure to the input sentence do not explicitly appear in the semantic representation. Instead the appropriate basic semantic function, in this case ANDER, is added to the semantic construction.

```
EX1:   2<3               result of scanner processing:
                         INTEGER ARITHREL INTEGER

STEP1:  (1,2)  S <- F
STEP2:  (2,1)    <- F2
STEP3:  (4,2)    <- EXPP ARITHREL EXPP
STEP4:  (7,1)    <- EXP1 ARITHREL EXP1
STEP5:  (13,1)   <- EXPT ARITHREL EXPT
STEP6:  (14,1)   <- EXPF ARITHREL EXPF
STEP7:  (15,2)   <- TERM ARITHREL TERM
STEP8:  (16,1)   <- INTEGER ARITHREL INTEGER
```

STEP1 tells us that we have an arithmetic formula (FML) and need to compute its truth value. The semantic construction at the end of STEP1 is:

(FML{ }).

STEP2 is the identity function. The semantic construction remains unchanged.

STEP3 is really the heart of the parse both syntactically and semantically. For each construction parsed there will be a step of this sort where each meaningful component will appear in as fully specified a form as necessary so that the correct semantic function can be assigned. Here it is the subset relation (S) which is important. {2} is a subset of the set of numbers that are less than 3. It would not actually be necessary at this level to know what the argument to the arithmetic relation is. That is, we could have written the rules:

```
F2        <- EXP ARITHRELS      (S ;1; ;2;)
ARITHRELS <- ARITHREL EXP       (APP (FCN ;1;) ;2;)
```

rather than

```
F2        <- EXP ARITHREL EXP   (S ;1; (APP (FCN ;2;) ;3;)).
```

[Note: FCN specifies that the argument given is a function name. For function names which are nouns, the FCN will be added to the semantic construction at the NP-level.]

The important point is that SUBSET is the basic semantic function for the construction and at this level it needs to locate both its arguments. It is unimportant how fully specified the arguments themselves are at this level; that decision can be made in terms of convenience. The semantic construction is now:

```
(FML (S[ ] (APP (FCN [ARITHREL])[ ]))).
```

> [Note: As explained above, the actual arithmetic function which in this case is the LESSTHAN function will be inserted by the macro-expander.]

STEP4 adds LST to the integer arguments. Like FCN, the LST function is used to give the evaluator information about the type of the argument. The semantic construction is now:

```
(FML (S (LST[ ]) (APP (FCN [ARITHREL]) (LST[ ])))).
```

STEPS5-7 are identities. The large number of uses of the identity function in this parse reflects the simplicity of the input. For example, in order to parse '2+3 < 3+3', the rule

```
(13,2)  EXP1 <- EXP1 + EXPT      (ADDER ;1; ;3;)
```

would be used instead of

```
(13,1)  EXP1 <- EXPT     ;1;
```

The use of the identity function allows us to drop down through levels of rules which are not needed for a given input.

STEP8 completes the syntactic parse and the semantic construction by specifying the two integers. The semantic construction is

```
(FML (S (LST [INTEGER]) (APP (FCN [ARITHREL]) (LST [INTEGER])))).
```

> [Note: this step by step analysis is not meant to be an accurate representation of any actual program implementation but rather a conceptual aid in understanding how the semantic functions work.]

EX2: 2<3, 3<4, and 4<5

```
result of scanner processing:
INTEGER ARITHREL INTEGER , INTEGER ARITHREL INTEGER ,
        AND INTEGER ARITHREL INTEGER
```

The only difference between the parse of this example and EX1 is the use of the listing feature. Instead of

```
STEP2:  (2,1)   <- F2                       ;1;
```

for this example we have:

```
STEP2:  (2,2)   <- LISTOFF                  (FML (ANDER (LST[ ])))
STEP2A: (3,4)   <- LISTOFF , /AND/ F2       (FML (ANDER (LST[ ][ ])))
STEP2B: (3,2)   <- F2 , F2 , /AND/ F2       (FML (ANDER (LST[ ][ ][ ])))
```

These rules provide an example of the use of semantic functions to handle correctly constructions involving unbounded branching of immediate constituents. The basic semantic function ANDER is inserted at STEP2 and then STEP2A and STEP2B simply pass the arguments up to it as they are parsed; thus preserving in the semantic construction the structure of the original input rather than adopting the artificial structure of the context-free parse.

One other feature of the semantic functions needs to be discussed here. If we have not yet written the semantic function for a particular rule, it will be marked UNDEFINED. However, some syntactic rules deliberately do not have semantic functions which is represented by the slot for the semantic function being blank. Blank semantics on a rule indicates one of two things. The word or phrase in question may be noise. Many noise words are eliminated in the scanner phase but there are words that have meaning in some contexts but are noise in others. For example, 'of the' in questions like 'Which of the factors of 12 are odd?' is a noise phrase. So we might write the rules:

```
1)  Q       <- INTER1 NP LINK SUBST      (1 ;2; ;4;)
2)  INTER1 <- INTER
3)  INTER1 <- INTER /OF/ /THE/
```

The semantic slot for rules 2 and 3 is blank. The INTER ('which' or 'what') has been taken into account at the level of rule 1, and if the phrase 'of the' occurs, it is noise.

However, blank semantics does not always indicate the presence of noise words. For example, consider the following questions:

How many tens does 50 equal?
How many tens does 50 name?
Does 50 name 5 tens?
Does 50 equal 5 tens?

In these contexts 'equal' and 'name' have the same meaning so we might write the rules:

1) Q <- /HOWMANY/ EXP AUX EXP VEQUAL
2) Q <- AUX EXP VEQUAL EXP
3) VEQUAL <- =
4) VEQUAL <- /NAME/

Any other verbs which have the same meaning in these contexts can be added to the category VEQUAL. 'Equal' and 'name' do not have the same meaning in every context, so we cannot associate them by means of either the TRANSL or the dictionary. However, the semantics of the construction is clear at the level of the Q-rules independent of which of the two verbs is used therefore the semantics for the VEQUAL-rules can be blank.

IV.2 S-Rules

(1,1)	S <- F .	(FML ;1;)
(1,2)	S <- F	(FML ;1;)
(1,3)	S <- Q ?	(QUS ;1;)
(1,4)	S <- C !	(CMD ;1;)
(1,5)	S <- D .	(DCL ;1;)
(1,6)	S <- D	(DCL ;1;)

These are the highest level rules. Their purpose is to determine what sort of sentence the input is: an arithmetic formula, a question, a command or a declarative. The period is optional for formulas and declaratives. Questions use a question mark and commands an exclamation point. I believe that the grammar can parse correctly without final punctuation if necessary for speech recognition. The semantic functions encode the type of the input sentence.

IV.3 F-Rules

```
(2,1)  F <- F2                          ;1;
(2,2)  F <- LISTOFF                     (ANDER (LST ;1;))

(3,1)  LISTOFF <- F2 /AND/ F2           ;1; ;3;
(3,2)  LISTOFF <- F2 , F2               ;1; ;3;
(3,3)  LISTOFF <- LISTOFF , F2          ;1; ;3;
(3,4)  LISTOFF <- LISTOFF , /AND/  F2   ;1; ;4;
(3,5)  LISTOFF <- LISTOFF /AND/ F2      ;1; ;3;

(4,1)  F2 <- EXPP ARITHREL EXPP         (ANDER (LST
                 ARITHREL EXPP          (S ;1; APP (FCN ;2;) ;3;))
                                        (S ;3;  (APP (FCN ;4;)  ;5;)))
(4,2)  F2 <- EXPP ARITHREL EXPP         (S ;1; (APP (FCN ;2;) ;3;))
```

The F-rules parse arithmetic formulas like:

2+2=4
2+2=3+1=4
2<3<4
{a,b} UNION {b,c} = {a,b,c}
1 lb. = 16 oz.
2 tens = 20 ones = 20 .

The rules also parse lists of these formulas. The evaluation program returns the truth-value. The semantic functions for the F-rules were discussed in Section IV.1.

IV.4 Top-Level EXP-Rules

```
(6,1)   EXP <- EXPCHOICE                         (CHL ;1;)
(6,2)   EXP <- LISTOFEXP                         (CHL ;1;)
(6,3)   EXP <- EXPP                              ;1;

(40,1)  EXPCHOICE <- EXPP , EXPP                 ;1; ;3;
(40,2)  EXPCHOICE <- EXPP /OR/ EXPP              ;1; ;3;
(40,3)  EXPCHOICE <- EXPCHOICE , EXPP            ;1; ;3;
(40,4)  EXPCHOICE <- EXPCHOICE , /OR/ EXPP       ;1; ;4;
(40,5)  EXPCHOICE <- EXPCHOICE /OR/ EXPP         ;1; ;3;

(12,1)  LISTOFEXP <- EXPP /AND/ EXPP             ;1; ;3;
(12,2)  LISTOFEXP <- EXPP , EXPP                 ;1; ;3;
(12,3)  LISTOFEXP <- LISTOFEXP , EXPP            ;1; ;3;
(12,4)  LISTOFEXP <- LISTOFEXP , /AND/ EXPP      ;1; ;4;
(12,5)  LISTOFEXP <- LISTOFEXP /AND/ EXPP        ;1; ;3;
```

LISTs have been a major problem. Initially, we believed that 'and' and 'or' should be given the same treatment as in formal logic. Thus a choicelist (disjunction) would have the value TRUE if the value for at least one member of the disjunction were TRUE and an ordinary list (conjunction) would be TRUE only if the values for all its members were TRUE. It immediately became obvious that this was not workable. Consider the following questions:

EX1: Is 2 or 3 even?
EX2: Are 2 and 3 even?

Using the standard logical interpretation, the answer to EX1 is TRUE and the answer to EX2 is FALSE. This is clearly unacceptable. Instead we have created the semantic function CHL (choicelist) which returns a separate answer for each member of the list. Given this output from the evaluator, it is a further question to decide exactly what information should be included in the final answer. Since CHL is assigned at the top -level in Rules (6,1) and (6,2), there is no need in this case to have Rules (40,1)-(40,5) as an exact duplication of (12,1)-(12,5). Instead they could be combined in a new category EXPLIST/CHOICE using the grammatical category AND/OR.

However the use of 'and' does not in every context call for this type of treatment. The following question:

EX3: Is {a} the intersection of {a b} and {a c}?

cannot use the CHL semantic function. The list in EX3 is a list of arguments to the INTERSECTION function which always requires more than one argument. The lexical category for these functions is 2FCN. The semantic function used for the list of arguments to a 2FCN is LST.

```
(87,9)  NP4 <- 2FCN /OF/ LST/EXP        (APP ;1; ;3;)
(88,2)  LST/EXP <- LISTOFEXP            (LST ;1;)
```

Since the arguments to a 2FCN are never listed using 'or', the LISTOFEXP-rules could not be combined with the EXPCHOICE-rules for this case.

The rule for ordinary FCN's is

```
(87,6)  NP4 <- FCN /OF/ NP              (APP ;1; ;3;)
```

If the argument is a list as in

EX4: What are the factors of 2, 3, and 4?

Rule (6,2) with the CHL semantic function will be used to further parse the NP. For EX3, LST will be assigned by rule (88,2). Thus LST or CHL is assigned at a higher level and then EX3 and EX4 will both use (12,1)-(12,5) to perform the parse of the actual list. Lists using the I and U semantic functions will be discussed in Section IV.14.

These recursive rules for lists need to be carefully written to prevent ambiguities. For example, the following rules

```
1)  A <- LISTOFA
2)  LISTOFA <- LISTOFA /AND/ A
3)  LISTOFA <- A /AND/ A
```

are unacceptable because by performing the recursion on A itself, elements of the list can jump back to the top level thereby creating sublists. Thus

A and A and A

can be parsed in the three following ways:

```
              A                      A                    A
              |                      |                    |
           LISTOFA                LISTOFA              LISTOFA
           /  |  \                /  |  \              /  |  \
    LISTOFA AND  A               A AND A              A AND A
     /  |  \                     |                          |
    A AND A                   LISTOFA                    LISTOFA
                               /  |  \                    /  |  \
                              A AND A                    A AND A
```

For this reason, in the LISTOFEXP and EXPCHOICE rules, the recursion is on EXPP rather than EXP.

IV.5 Types of EXP's

```
(7,1)  EXPP <- EXP1              (LST ;1;)
(7,2)  EXPP <- SETEXP            ;1;
(7,3)  EXPP <- DATEXP            ;1;
(7,4)  EXPP <- NTUPLE            ;1;
(7,5)  EXPP <- TIMEXP            ;1;
(7,6)  EXPP <- ARITHEXP          (LST ;1;)
```

We have divided noun phrases into two categories. The NP1 category parses the more flexible natural language noun phrases and the

EXP-rules parse the phrases which have a predefined format in mathematical language. Thus all the rules for parsing constructions peculiar to this subject matter are segregated in one portion of the grammar which makes the grammar more readable. EXP is the category which could be eliminated if the subject matter were changed and arithmetic was not needed. It is also the category which will need the most work as new topics in elementary mathematics are added to the scope of the program.

The different types of EXP's will be discussed in the next sections. Only six categories of EXP's are given in the EXPP rules listed at the beginning of this section. There are in fact other equally distinguishable categories which appear in the TERM rules below rather than here with the EXPP's. One example of such a category is UNITS (expressions of units of measurement). These categories need to be TERM's because they can function as terms in arithmetic expressions like

2 feet + 4 feet = 2 yards.

IV.6 EXP1-Rules

```
(13,1)  EXP1 <- EXPT                        ;1;
(13,2)  EXP1 <- EXP1 + EXPT                 (ADDER ;1; ;3;)
(13,3)  EXP1 <- EXP1 - EXPT                 (SUBBER ;1; ;3;)

(14,1)  EXPT <- EXPF                        ;1;
(14,2)  EXPT <- EXPT * EXPF                 (MULTER ;1; ;3;)
(14,3)  EXPT <- EXPT / EXPF                 (DIV ;1; ;3;)

(15,1)  EXPF <- ( EXP1 )                    ;2;
(15,2)  EXPF <- TERM                        ;1;

(16,1)  TERM <- INTEGER                     ;1;
```

```
(16,2)  TERM <- REAL                            ;1;
(16,3)  TERM <- RNUMERAL                        ;1;
(16,4)  TERM <- INTEGER INTEGER / INTEGER       (MAKEMIXED ;1; ;2; ;4;)
(16,5)  TERM <- VAR                             ;1;
(16,6)  TERM <- CARDOF SETEXP                   (CARDINALITY ;2;)
( 6,7)  TERM <- NUNITS                          ;1;
(16,8)  TERM <- UNITS                           ;1;
(16,9)  TERM <- /NUMBER/ /PERCENT/              (PERCENT ;1;)
(16,10) TERM <- /NUMBER/ /PERCENT/ /OF/ EXP     (PERCENTAGE ;1; ;4;)
(16,12) TERM <- INTEGER /OF/ INTEGER /PART/     (DIV ;1; ;3;)
(16,13) TERM <- INTEGER /OUTOF/ INTEGER         (DIV ;1; ;3;)

(17,1)  /NUMBER/ <- INTEGER                     ;1;
(17,2)  /NUMBER/ <- REAL                        ;1;
(17,3)  /NUMBER/ <- INTEGER / INTEGER           (DIV ;1; ;3;)
(17,4)  /NUMBER/ <- INTEGER INTEGER / INTEGER   (MAKEMIXED ;1; ;2; ;4;)
```

These rules are standard rules for parsing arithmetic expressions. The substance of the rules was taken from [7]. The rules are designed to reflect the ordinary precedence rules for arithmetic operators.. The category /NUMBER/ is defined for use in rules such as (16,9) and (16,10) where only single numbers (integers, decimals, mixed numbers, and fractions) can appear rather than arithmetic expressions like 2+4.

The scanner recognizes integers and reals but the current semantic functions ignore this information since no evaluation procedures have yet been written that are specifically designed for reals. The evaluation program uses predefined LISP functions whenever possible. The standard LISP functions for arithmetic work well for operations on integers, but the evaluator should not use them for operations on reals. LISP uses floating point for reals thus causing the answer 3.999... to be given to the question 'What is the sum of 1 and 3.0?'. This is one of the areas where the evaluation program simply has not yet been completed. A procedure needs to be written which does decimal arithmetic in the same way as an elementary student rather than the same way as a computer. Note that when the evaluation program is ready to deal with the information that a number is a real, it will be given this information through the semantic functions on rules (16,1), (16,2), (17,1), and (17,2).

Rule (16,4) parses mixed numbers. Mixed numbers can be identified by the grammar, but unfortunately no way has been implemented to distinguish fractions from ordinary division. This problem is created by the limited character set of the teletype. We have chosen to use '/' for both division and fractions rather than to designate an arbitrary character for one of these purposes. The representation of fractions as a division operation requires special techniques for their handling in the evaluation program. Rule (16,6) derives the expression for cardinality of sets, for example, n [a b c] = 3. Rules (16,7) and (16,8) parse expressions involving percents and (16,12) parses expressions like '36 of 100 parts'. Of course, EXP1 will derive each of these types of TERM's singly as well as combined by the arithmetic operators.

IV.7 Set-Expressios and Ntuples

```
(21,1)  SETEXP <- SETEXP /UNION/ SETEXP          (U ;1; ;3;)
(21,2)  SETEXP <- SETEXP /INTER/ SETEXP          (I ;1; ;3;)
(21,3)  SETEXP <- SETEXP - SETEXP                (SD ;1; ;3;)
(21,4)  SETEXP <- { }                            NIL
(21,5)  SETEXP <- { ELELIST }                    (LST ;2;)
(21,6)  SETEXP <- { ELELIST1 }                   (LST ;2;)
```

```
(21,7)  SETEXP <- { VAR : F }                    (ABSTRACT ;2; (;4;))
(21,8)  SETEXP <- { VARA : F }                   (ABSTRACT ;2; (;4;))

(22,1)  ELELIST <- ELEMENT , ELEMENT             ;1; ;3;
(22,2)  ELELIST <- ELELIST , ELEMENT             ;1; ;3;

(23,1)  ELELIST1 <- ELEMENT                      ;1;
(23,2)  ELELIST1 <- ELELIST1 ELEMENT             ;1; ;2;

(24,1)  ELEMENT <- PN                            ;1;
(24,2)  ELEMENT <- EXP1                          ;1;
(24,3)  ELEMENT <- SETEXP                        ;1;

(11,1)  NTUPLE <- < ELELIST >                    (NTUPLE ;2;)
```

The evaluation procedures for NTUPLEs and SETEXPs have not been worked on extensively yet. The rules and semantic functions which appear here need some revision. For example, precedence rules will be needed like those for arithmetic operators.

The elements of a set are parsed by ELELIST if they are separated by commas and ELELIST1 if they are not. To avoid ambiguity, the singleton set is parsed only by ELELIST1. We have not yet decided what sorts of objects should be allowed as elements of sets. Clearly SETEXP's, /NUMBER/'s, and VAR's should be allowed but the category EXP1 is probably not too broad. Proper names (PN) would be desirable since they are often used in examples in elementary textbooks, but they present a problem with the dictionary. Common nouns that are frequently used in sets, for example, {2 apples, 3 banannas, 5 pears} are also not in the dictionary and would be hard to distinguish from nouns which are not likely to be so used.

IV.8 DATEXP and TIMEXP-Rules

```
(10,1)  TIMEXP <- INTEGER : INTEGER APM        (TIMER ;1; ;3; ;4;)
(10,2)  TIMEXP <- INTEGER APM                  (TIMER ;1; 0 ;2;)
(10,3)  TIMEXP <- INTEGER /O'CLOCK/ APM        (TIMER ;1; 0 ;3;)
(10,4)  TIMEXP <- INTEGER /O'CLOCK/            (TIMER ;1; 0 @AM)

(38,1)  DATEXP <- MONTH INTEGER                (DATE (UNT ;1; ;2;))
(38,2)  DATEXP <- MONTH INTEGER , INTEGER      (DATE (UNT ;1; ;2; ;4;))
```

Again, no work has been done on the evaluation of these expressions, but these are the rules which will parse dates and times. Depending on the method both syntactic and semantic that is adopted for dealing with calendar-type questions the DATEXP rules may need to be moved to another portion of the grammar. Mathematical questions dealing with time appear to be either requests for conversion (for example, from 24 to 12-hour time) or comparisons (for example, 'Which is earlier?'). These function like other units of measurement except that they cannot be TERM's in arithmetic expressions.

IV.9 ARITHEXP-Rules

```
(9,1)  ARITHEXP <- NP /DIVIDEDBY/ NP            (DIV ;1; ;3;)
(9,2)  ARITHEXP <- NP /ADDEDTO/ NP              (ADDER ;1; ;3;)
(9,3)  ARITHEXP <- NP /SUBTRACTEDFROM/ NP       (SUBBER ;3; ;1;)
(9,4)  ARITHEXP <- NP /MULTIPLIEDBY/ NP         (MULTER ;1; ;3;)
(9,5)  ARITHEXP <- NP /DIVIDEDINTO/ NP          (DIV ;3; ;1;)
```

The ARITHEXP rules parse arithmetic expressions in which the operator is written out as a verb with a preposition. The TRANSL portion of the scanner is used to convert 'plus', 'minus', and 'times' to '+', '-', and '*'. 'Divided by', 'multiplied by', and 'added to' could also be handled by the TRANSL, but 'subtracted from' and 'divided into' cannot since they need their arguments reversed in order to use the regular subtraction and division semantic functions. The scanner could be made to perform the conversion including the reversal of the arguments, but this does not seem to be a natural function to expect the scanner to perform. All five ARITHEXP's have been included in the grammar to maintain uniformity of treatment.

IV.10 UNIT and NUNIT-Rules

```
(25,1)  NUNITS <- JOINNUNITS                    (NUNITJOIN (NUNT ;1;))
(25,2)  NUNITS <- SUMNUNITS                     (NUNITJOIN (NUNT ;1;))

(26,1)  UNITS <- JOINUNITS                      (UNITJOIN (UNT ;1;))
(26,2)  UNITS <- SUMUNITS                       (UNITJOIN (UNT ;1;))

(27,1)  JOINUNITS <- UNIT1                      ;1;
(27,2)  JOINUNITS <- JOINUNITS UNIT1            ;1; ;2;

(28,1)  JOINNUNITS <- NUNIT1                    ;1;
(28,2)  JOINNUNITS <- JOINNUNITS NUNIT1         ;1; ;2;

(29,1)  SUMNUNITS <- NUNIT1 /AND/ NUNIT1        ;1; ;3;
(29,2)  SUMNUNITS <- SUMNUNITS , NUNIT1         ;1; ;3;
(29,3)  SUMNUNITS <- NUNIT1 , NUNIT1            ;1; ;3;
(29,4)  SUMNUNITS <- SUMNUNITS , /AND/ NUNIT1 ;1; ;4;
(29,5)  SUMNUNITS <- SUMNUNITS /AND/ NUNIT1     ;1; ;3;

(30,1)  SUMUNITS <- UNIT1 /AND/ UNIT1           ;1; ;3;
(30,2)  SUMUNITS <- SUMUNITS , UNIT1            ;1; ;3;
(30,3)  SUMUNITS <- UNIT1 , UNIT1               ;1; ;3;
(30,4)  SUMUNITS <- SUMUNITS , /AND/ UNIT1      ;1; ;4;
(30,5)  SUMUNITS <- SUMUNITS /AND/ UNIT1        ;1; ;3;

(31,1)  UNIT1 <- /NUMBER/ /OPER/ UNIT           (;1; (;2; ;3;))
(31,2)  UNIT1 <- /A/ /OPER/ UNIT                (MAKEUNIT (1 (;2; ;3;)))
(31,3)  UNIT1 <- /NUMBER/ UNIT                  (;1; (;2;))
(31,4)  UNIT1 <- /a/ UNIT                       (1 (;2;))

(32,1)  NUNIT1 <- INTEGER NUNIT                 (;1; (;2;))
(32,2)  NUNIT1 <- DET NUNIT                     (1 (;2;))
```

Again the evaluation procedures for these have not yet been written so the rules have not been extensively tested. I have created the category NUNITS for ones, tens, hundreds, etc. Their evaluation is sufficiently different from other UNITs (like inches, teaspoons, etc.), that this distinction which can easily be made by the grammar provides useful information to the evaluation program.

These rules were discussed in Section III.4 as an example of the handling of the ambiguity problem. The rules as they appear here will produce all of the derivations discussed in Section III.4 but, unfortunately, they will also produce a few unacceptable ambiguities. The expression

5 yds., 2 ft., and 3 in. and 4 yds., 2 ft., and 5 in.

is a two-element list that needs to be parsed by LISTOFEXP with the two elements on the list each parsed by SUMUNITS which forms lists of units into a single compound unit. However, allowing sublists of the LISTOFEXP expression to be parsed by SUMUNITS has catastrophic results. The way the rules now stand SUMUNITS can pick off almost any sublist in addition to the ones that it is supposed to handle. As the lists become longer the ambiguities multiply rapidly. These rules need very careful rewriting before they will work properly.

Single units will be parsed by JOINUNITS. A single unit, either alone or as an element of a list, must always consist of at least two parts -- the type of the unit and the number, for example, '3 feet'. Unit names alone are not parsed by these rules. The determiner

/A/ in, for example, 'a foot equals 12 inches' means one so the '1' has been inserted in the semantic function. The category /OPER/ contains the words 'square', 'cubic', etc. This method was chosen in preference to our original method of using the TRANSL to join 'square' and 'cubic' to the unit name. Due to abbreviations of the two words with and without periods, 'square feet' alone took twelve entries in the TRANSL. The current method also reflects the more proper approach to the evaluation of the construction.

IV.11 Geometric Measurements

```
(33,1)  UNITLIST3 <- UNITS MEASWORD
(33,2)  UNITLIST3 <- UNITLIST3 , UNITS MEASWORD
(33,3)  UNITLIST3 <- UNITLIST3 /AND/ UNITS MEASWORD
(33,4)  UNITLIST3 <- UNITLIST3 , /AND/ UNITS MEASWORD

(34,1)  UNITLIST4 <- UNITS /BY/ UNITS
(34,2)  UNITLIST4 <- UNITLIST4 /BY/ UNITS
(34,3)  UNITLIST4 <- UNITS MEASWORD
(34,4)  UNITLIST4 <- UNITLIST4 /BY/ UNITS MEASWORD

(35,1)  UNITLIST5 <- UNITS
(35,2)  UNITLIST5 <- /NUMBER/ /BY/ /NUMBER/ UNIT
(35,3)  UNITLIST5 <- /NUMBER/ /BY/ /NUMBER/ /BY/ /NUMBER/ UNIT

(36,1)  UNITLIST6 <- /3D/ UNITS
(36,2)  UNITLIST6 <- /3D/ UNITS /AND/ /3D/ UNITS
(36,3)  UNITLIST6 <- /3D/ UNITS , /3D/ UNITS , /AND/ /3D/ UNITS
(36,4)  UNITLIST6 <- /3D/ UNITS , /3D/ UNITS /AND/ /3D/ UNITS

(37,1)  UNITLISTS <- UNITLIST3
(37,2)  UNITLISTS <- UNITLIST4
(37,3)  UNITLISTS <- UNITLIST5
(37,4)  UNITLISTS <- UNITLIST6
```

The semantic functions for all of these rules are currently UNDEFINED. The UNITLIST rules parse the measurements of geometric figures. These measurements can be given a) in an adjectival form ('a 2 X 3 inch rectangle'), b) in a relative clause ('a rectangle whose dimensions are length 3 inches and width 2 inches'), or c) in an appositive position ('a rectangle, 2 inches wide and 3 inches long'). The noun phrase rules for these three cases are:

```
a)  NP6 <- UNITLISTS GEOFIGURE
b)  NP6 <- GEOFIGURE RELPOSPRONS
c)  NP6 <- GEOFIGURE , UNITLISTS ,
```

GEOFIGURE is the lexical category for nouns like 'square' and 'rectangle'. The purpose of the semantic functions for the UNITLIST rules will be to put the measurements and their associated dimensions into a standardized format regardless of the format of the input. The measurements will then be attached to the name of the geometric figure at the NP6 level.

IV.12 Relative Clauses

```
(52,1)  RELPOSPRONS <- RELPRON RLPR                       ;2;
(52,2)  RELPOSPRONS <- RELPOS RLPS                        ;2;
(52,3)  RELPOSPRONS <- RELPRON RLPR /AND/ RELPRON RLPR  (1 ;2; ;5;)
(52,8)  RELPOSPRONS <- RELPRON RLPR /AND/ RLPR          (1 ;2; ;4;)

(54,1)  RLPR <- LINK SUBST/EXP                            ;2;
(54,2)  RLPR <- LINK UNITLISTS                            ;2;
(54,3)  RLPR <- /HAVE/ /DIMENSIONS/ FUNCHOICE UNITLISTS UNDEFINFD
(54,4)  RLPR <- = NP                                      ;2;
(54,5)  RLPR <- /HAVE/ UNITLIST6                          UNDEFINED
(54,6)  RLPR <- /HAVE/ /DIMENSIONS/ FUNCHOICE LISTOFEXP UNDEFINED
(54,7)  RLPR <- /HAVE/ HAVENP                             ;2;
(54,8)  RLPR <- ENPAS                                     ;1;
(54,9)  RLPR <- /HAVE/ COMPHENP                           UNDEFINED
```

The relative pronouns (RELPRON) are 'that' and 'which' and the relative possessive (RELPOS) is 'whose'. The relative clauses using a relative pronoun have been fully implemented (except for those involving units), but the relative possessives which are much more complicated semantically have not yet been implemented and may be disregarded here. Their treatment will be similar to that for HAVENP's which they strongly resemble. The semantic function for noun phrases containing a relative clause is intersection. For example, 'the factors of 12 that are prime numbers' are found by intersecting the set of factors of 12 with the set of prime numbers.

I will give an example for each of the RELPOSPRONS-rules that use relative pronouns:

(52,1) that are less than 5
(52,2) that are less than 5 and that are greater than 2
(52,8) that are less than 5 and are greater than 2.

[Note: 'that are less than 5 and greater than 2' will be parsed by (52,1) in combination with (54,1) because 'less than 5 and greater than 2' is parsed by SUBST.]

These rules only allow two elements in a list. If this is found to be inadequate, recursive listing rules can easily be written. Phrases of the form RELPRON RLPR RELPRON RLPR ('an even number that is less than 5 that is greater than 2') will be parsed by successive applications of RELPOSPRONS in the NP rules.

Rules (54,2), (54,3), (54,5) and (54,6) are used for relative clauses which give geometric measurements (see Section IV.11 above). Rule (54,1) is the most common form of relative clause. The SUBST/EXP may be any one of the following:

(1) a noun phrase:
'that is a prime number'
(2) an adjective:
'that is even'
(3) a prepositional phrase:
'that is between 5 and 10'
(4) a unit-conversion phrase:
'that is in lowest terms'
(5) an arithmetic relation:
'that is less than 5'
(6) an EXP:
'that is 2% of 20'

Examples of Rules (54,4) and (54,7)-(54,9) are:

(54,4) an improper fraction that equals the whole number 2!
(54,7) a fraction that has 1 as denominator
(54,8) the odd number that 12 has as a factor
(54,9) numbers that have more than 2 factors

IV.13 Prepositions

```
(55,1)  PREPHRASE1 <- /BETWEEN/ NP /AND/ NP        (BETWEEN ;2; ;4;)
(55,2)  PREPHRASE1 <- /BEFORE/ NP                  (BEFORE ;2;)
(55,3)  PREPHRASE1 <- /AFTER/ NP                   (AFTER ;2;)
(55,4)  PREPHRASE1 <- /IN/ EXP                     UNDEFINED
(55,5)  PREPHRASE1 <- /IN/ DET APPOSN EXP          UNDEFINED

(56,1)  PREPHRASE <- PREPHRASE1 /OR/ PREPHRASE1    (CHL ;1; ;3;)
(56,2)  PREPHRASE <- PREPHRASE1 /AND/ PREPHRASE1   (I ;1; ;3;)
(56,3)  PREPHRASE <- PREPHRASE1                    ;1;
(56,4)  PREPHRASE <- /NOT/ PREPHRASE1              (C ;2;)
(56,5)  PREPHRASE <- /NEITHER/ PREPHRASE1 /NOR/ PREPHRASE1
                                                   (NOR ;2; ;4;)
(56,6)  PREPHRASE <- /EITHER/ PREPHRASE1 /OR/ PREPHRASE1
                                                   (U ;2; ;4;)
```

I studied the use of prepositions in the entire corpus of [23], both in the questions and in the exposition. I discovered that the use of prepositions in the exposition was significantly broader than the use in the questions. The number of prepositions and the variety of use of each is sufficiently small in questions to be manageable in the first stage of our program. Certain preposition uses like the use of 'by' and 'to' in 'Count by fives to 100!' are integrally related to the verb. I will discuss these in Section IV.22. Another common use of prepositions in elementary mathematics is to indicate ordering, for example, 'in order from largest to smallest'. The ORDERING rules will be discussed in Section IV.21. And the use of prepositions in the expression of arithmetic operations like 'added to' and 'divided by' was discussed in Section IV.9.

I found fourteen prepositions used in the complete corpus of [23]. Of these the four least frequently used prepositions (about, on, over, and without) have not been included in our grammar. Also, certain infrequent uses of another six prepositions (by, for, from, in, into , and to) have not been included, but the majority of uses of these six have been included. And the remaining four prepositions (as, between, of and with) have been completely implemented. I will discuss each one of the fourteen prepositions and indicate which uses of it we have implemented.

1) About has not been implemented. It was found only in the contexts 'talking about' and 'asked about'.

2) As is used in conversions, for example, 'Express .04 as a percent!' and will be discussed in Section IV.18.

3) Between was found to have only the mathematical meaning of a number being between two other numbers.

4) By is used in the context of arithmetic operations

('multiplied by'), with other verbs ('count by'), and in giving geometric measurements ('2 by 4 in.'). Also certain instances of 'by' are handled by the TRANSL file. 'Divisible by' is TRANSL'd to DIVBY which is an arithmetic relation. 'Divisible by 5' creates a set in the same way that 'greater than 5' does. I have also TRANSL'd 'can be divided by' to 'is DIVBY'. 'By size' is TRANSL'd to 'in order'. There were other uses of 'by' like the following:

a) Check by using the inverse operation
b) We can check subtraction by addition
c) Find an equal fraction by multiplying the numerator and denominator...
d) Solve each equation by rewriting it as an equation using division.

There seems to be a common pattern in these examples, but we have not yet implemented this use of 'by'.

5) For is being treated as equivalent to 'of' except in the phrase 'except for' which is TRANSL'd. Some examples of this use are:

1) Write the simplest name for 300+40+6!
2) Find the answer set for {3,4,5} - {4,5}!
3) What is the least common denominator for 1/4 and 5/6?
4) Find a solution for the equation 5+X = 10!

The semantic function for 'of' is (APP ;1; ;3;). For example, 'the factors of 12' are found by applying the FACTOR function to 12. Thus, 'the solution for the equation 5+x=10' will be found by applying the procedure for solving equations to the given equation. Another use of 'for' in mathematical contexts which will have to be considered is the use of 'for' in phrases like 'for any number N'. We might also write a rule for the following format which is used frequently in [23]:

We write: in. for inch or inches.

Other uses of 'for' were found in [23]. Some of the phrases were:

1) A reason for
2) Distributive law for multiplication over addition
3) Standard unit of measure for area
4) For many purposes
5) Symbol for zero

6) From is used in ORDERINGs. It is also used with the verb 'convert' as in 'Convert 3/5 from a fraction to a decimal!'. The only other use found in [23] was with the verb 'obtain' and this has not been implemented.

7) In is a very common preposition. The following are the uses which have been implemented:

a) The rule NP1 <- /THE/ EXP1 /IN/ EXP1 has been written for expressions like 'the 9 in 891'. This expression evaluates to 9 tens which seems to be the intended meaning in the text where questions like 'What is the 9 in 891?' are asked.

b) 'In' is used in requests for conversion, for example 'Write 42 in Roman numerals!' and 'express the answer in cubic inches!'

c) The two expressions 'in lowest terms' and 'in expanded form' appear so frequently that we have TRANSL'd them to single words and made them terminal categories in the grammar.

d) 'In' is often used to mean membership. Two examples are 'Is 8 in [6,7,8]?' and 'Rename a fraction in the pair...'.

e) 'In order' as in 'list in order' is TRANSL'd to a single word and used in ORDERINGs.

f) Expressions like 'earlier in the day' are TRANSL'd to 'earlier' since 'in the day' provides no needed information to the evaluator.

g) 'In' is used to state or request measurements, for example, 'How many days are there in September?' and 'There are 12 inches in a foot.'

There are other uses of 'in'. We would like to find a common method of dealing with at least some of the seven uses above, as well as many which have not yet been dealt with at all.

8) Into is used with the verb 'divide'. Examples like the following were found in [23]:

> a) If we divide a set of things into two sets having the same number of things, each of the small sets is one half of the whole set.
>
> b) When we divide something into three parts the same size, each part is one third of the whole thing.
>
> c) If we divide a set of things into thirds, we make three small sets.

This use of 'divide into' has not been implemented, but we have included rules which handle 'divide into' when it is used for ordinary division, for example, 'Divide 7 into 56!'.

9) Of is the most commonly used preposition in the context of elementary mathematics. It is used for specifying functions and their arguments, for example, 'factors of 12', 'union of {a} and {b}', 'sum of 2 and 3', 'subset of {a b}', 'set of numbers less than 3', 'numerator of 2/3', 'area of a 2 inch square', 'number of days in September', and 'member of { b}'.

10) On has not been implemented in this grammar. It was found in phrases such as: 'Show 4+2 on a number line', 'perform the union operation on...', 'the operations on sets', 'on each side', and 'on a thermometer'.

11) Over was found only in specifications of the distributive laws, for example, 'multiplication over division'.

12) To appears with the verbs: count, equal, round, change, and convert. It is also used in ordering expressions, e.g., 'from largest to smallest'. There were two other phrases that we have not dealt with, 'common to' and 'to the right of'.

13) With is TRANSL'd to 'that has'. Some examples are:

a) The set with no things in it...
b) The volume of a box with length 7 inches...
c) Fractions with the same denominator...
d) A number with exactly two factors...

When other uses of 'with' (note that no others were found in [23]) are implemented it will no longer be practical to use the TRANSL in this way. Hopefully, there will be a clear-cut grammatical way to discriminate between the uses.

14) Without was found in the sentence, 'We can multiply the dividend and divisor by the same number without changing the value of the expression.'

The prepositions 'before' and 'after' did not appear in [23] but we have implemented them with the meanings successor and predecessor, for example, '2 comes before 3'.

Of all these prepositions, only 'between', 'before', 'after', and 'in' (in the sense of membership) are included in the category PREPHRASE given at the beginning of this section. Sets can be constructed by the evaluator from these prepositional phrases, for example, the set of numbers between five and ten. PREPHRASE's are one of the types of SUBST's.

IV.14 SUBST-Rules

```
(58,1)  SUBST <- SUBST1                             ;1;
(58,2)  SUBST <- SUBST1 /AND/ SUBST1                (I ;1; ;3;)
(58,3)  SUBST <- SUBST1 /OR/ SUBST1                 (CHL ;1; ;3;)
(58,4)  SUBST <- SUBST1 , SUBST1 , /AND/ SUBST1     (IER ;1; ;3; ;6;)
(58,5)  SUBST <- SUBST1 , SUBST1 /AND/ SUBST1       (IER ;1; ;3; ;5;)
(58,6)  SUBST <- SUBST1 , SUBST1 , /OR/ SUBST1      (CHL ;1; ;3; ;6;)
(58,7)  SUBST <- SUBST1 , SUBST1 /OR/ SUBST1        (CHL ;1; ;3; ;5;)
(58,8)  SUBST <- /NEITHER/ SUBST1 /NOR/ SUBST1      (NOR ;2; ;4;)
(58,9)  SUBST <- /EITHER/ SUBST1 /OR/ SUBST1        (U ;2; ;4;)

(59,1)  SUBST1 <- NP1                               ;1;
(59,2)  SUBST1 <- NP3                               ;1;
(59,3)  SUBST1 <- ADJ                               ;1;
(59,4)  SUBST1 <- PREPHRASE1                        ;1;
(59,5)  SUBST1 <- ARITHREL NPVARA                   (APP (FCN ;1;) ;2;)
(59,6)  SUBST1 <- SPECPREP1                         ;1;
(59,7)  SUBST1 <- V SPECPREP1                       ;2;
(59,8)  SUBST1 <- /NOT/ SUBST1                      (C ;2;)
```

I will give an example of each of the types of SUBST's given in rules (59,1)-(59,8).

(59,1) Is 5 an odd number?

(59,2) Are 5 and 7 odd numbers?

> Note: The need for two rules here in order to parse all the SUBST's which are noun phrases is caused by the failure to distinguish singular from plural.

(59,3) Is 5 odd?

(59,4) Is 5 between 1 and 10?

(59,5) Is 5 less than 10?

(59,6) Is 2/5 in lowest terms?

(59,7) Is 1/2 expressed as a fraction, a decimal, or a percent?

(59,8) Is 5 even or not even?

The category used in (59,4) is PREPHRASE1 rather than PREPHRASE. In this as well as the other SUBST-rules, the substantive element is taken at a level which does not allow listing of the elements or the complement of the element. Thus the list and complement contained in the question 'Is 5 odd and not between 1 and 5?' will both be handled by the top-level SUBST rules. This is necessary when the elements of the list of SUBST's are not from the same grammatical category. If PREPHRASE rather than PREPHRASE1 were used, the list in 'Is 8 between 5 and 10 and after 7?' could be parsed by either the PREPHRASE or the SUBST rules and thus would be ambiguous.

Rules (58,1)-(58,9) parse lists of SUBST's. In Section IV.4 above I discussed the use of CHL and LST as semantic functions for lists. CHL can be used for lists and choicests of SUBST's. For example, if CHL is used, the answer to

EX1: Is 2 a factor of 2 and also a multiple of 2?

will be

(QUS (CHL (TV T) (TV T))).

There is, however, another approach which can be taken for lists. In this approach I (intersection) is used for lists and U (union) for choicelists. Thus EX1 would be interpreted as meaning

Is 2 in {x | x is a factor of 2} INTERSECTION
{x | x is a multiple of 2} .

This use of the set-theoretical functions I and U is similar to the logical approach suggested in Section IV.4 and has the disadvantage discussed in that section of not providing a complete enough specification of the answer for yes/no questions like EX1. However, there are constructions involving lists which do require the I or U functions.

EX2: Give a number that is less than 6 and greater than 2!
EX3: Is any number divisible by 6 and not divisible by 3!

The rule we have been using for questions like EX1 and EX3 is

RULE1: Q _ LINK NP SUBST (S ;1; ;3;)

This rule is too general. The specific determiner used for the NP should determine the semantic function to be used at the level of RULE1. The existential quantifier, as in EX3, will require the I function rather than the S (subset) function. Since EX2 contains the list in a relative clause which also uses the I function, we might form the hypothesis that constructions using the I function should use I or U for any lists contained in the construction and similarly, if the semantic function is S, lists contained in the construction should use CHL. In order to implement this hypothesis, we could create the two categories CHLSUBST and I/USUBST which would be used at the level of RULE1 instead of the current category SUBST. This change has not yet been made for two reasons. First the determiners which give us the information about which semantic function should be used for rules like RULE1 have not yet been worked out. And second, a more serious problem is that this approach really does not work satisfactorily.

EX4: Is 2 a factor of 2 or 3?
EX5: Is any number that is a factor of 2 or 3 also
a factor of any other prime number?

In these examples, the list '2 or 3' is buried several levels down in the grammar. In EX4 the levels are SUBST and FCN. In EX5, they are SUBST, RELPRONS, and FCN. To implement the suggested approach, a pair of grammatical categories would be needed at each level in order to carry down the information as to whether CHL or I/U were needed. This would result in an unnecessarily complicated grammar. Problems of this sort are much more efficiently handled by the semantic functions which are more flexible and more powerful than the grammar. The evaluator works inside-out. A semantic function needs to be created which will postpone evaluation of the list until the appropriate time when the information is known as to which function to use. Until this can be implemented, we have assigned a semantic function to each of the SUBST-rules and other listing rules which reflects the most common case for the particular rule.

IV.15 Arithmetic Relations

```
(74,1)  ARITHRELS <- NOT/ARITH                                 ;1;
(74,2)  ARITHRELS <- ARITHRELS , NOT/ARITH                     (I ;1; ;3;)
(74,3)  ARITHRELS <- ARITHRELS , /AND/ NOT/ARITH               (I ;1; ;4;)
(74,4)  ARITHRELS <- ARITHRELS , /OR/ NOT/ARITH                (CHL ;1; ;4;)
(74,5)  ARITHRELS <- NOT/ARITH /OR/ NOT/ARITH                  (CHL ;1; ;3;)
(74,6)  ARITHRELS <- /EITHER/ NOT/ARITH /OR/ NOT/ARITH         (CHL ;2; ;4;)
(74,7)  ARITHRELS <- /NEITHER/ NOT/ARITH /NOR/ NOT/ARITH       (NOR ;2; ;4;)

(76,1)  NOT/ARITH <- ARITHREL NPVARA             (APP (FCN ;1;) ;2;)
(76,2)  NOT/ARITH <- /NOT/ ARITHREL NPVARA       (C (APP (FCN ;2;) ;3;)
```

[Note: For a discussion of the arithmetic relations see page 30.]

IV.16 Adjective Rules

```
(80,1)  ADP <- /NEITHER/ ADP /NOR/ ADP          (NOR ;2; ;4;)
(80,2)  ADP <- /EITHER/ ADP /OR/ ADP            (CHL ;2; ;4;)
(80,3)  ADP <- ADP2                             ;1;
(80,4)  ADP <- /NOT/ ADP2                       (C ;2;)
(80,5)  ADP <- ADJANDOR                         (CHL ;1;)
(80,6)  ADP <- ADJANDOR ADP2                    (CHL ;1; ;2;)

(81,1)  ADP2 <- ADJ                             (STS ;1;)
(81,2)  ADP2 <- ADP2 ADJ                        (I ;1; (STS ;2;))
(81,3)  ADP2 <- ADP2 , ADJ                      (I ;1; (STS ;2;))

(82,1)  ADJANDOR <- ADJ /OR/ ADJ                (STS ;1;) (STS ;3;)
(82,2)  ADJANDOR <- ADJ /AND/ ADJ               (STS ;1;) (STS ;3;)
```

[Note: For a discussion of adjectives see page 29].

IV.17 CONVUNITS-Rules

```
(67,1)  CONVUNITS <- /NEITHER/ CONVUNITS /NOR/ CONVUNITS
(67,2)  CONVUNITS <- /EITHER/ CONVUNITS /OR/ CONVUNITS
(67,3)  CONVUNITS <- LISTNAMESU
(67,4)  CONVUNITS <- LISTNAMESNU
(67,5)  CONVUNITS <- LISTOFCONVUNITS
(67,6)  CONVUNITS <- CONVUNITSCHOICE
(67,7)  CONVUNITS <- CONVUNITS1
(67,8)  CONVUNITS <- DET CONVUNITS1

(68,1)  CONVUNITS1 <- N
(68,2)  CONVUNITS1 <- N RELPOSPRONS

(104,1)  LISTNAMESU <- UNITT
(104,2)  LISTNAMESU <- LISTNAMESU /AND/ UNITT
(104,3)  LISTNAMESU <- LISTNAMESU , UNITT
(104,4)  LISTNAMESU <- LISTNAMESU , UNITT , /AND/ UNITT

(105,1)  LISTNAMESNU <- NUNIT
(105,2)  LISTNAMESNU <- LISTNAMESNU /AND/ NUNIT
(105,3)  LISTNAMESNU <- LISTNAMESNU , NUNIT
(105,4)  LISTNAMESNU <- LISTNAMESNU , NUNIT , /AND/ NUNIT
```

Many questions and especially commands in elementary mathematics call for conversion of an expression from one form to another, for example,

EX1: Give 1/4 as a decimal and as a percent!

The rules in this section deal with the forms being converted to, which in EX1 are 'decimal' and 'percent'. The semantic functions for all of these rules are currently UNDEFINED. The CONVUNITS rules parse lists and choicelists of form names and also rule (67,8) allows an optional determiner before the form name. There are several lexical categories of nouns. The relevant noun categories for these rules are UNIT, NUNIT, and N. UNITs and NUNITs were discussed in Section IV.10. N's are nouns which name sets represented by a characteristic function. Examples are number, fraction, decimal, Roman numeral, and whole number. Sometimes the N will be modified by a relative clause (rule (68,2)) as in 'Give .10 as a fraction whose denominator is 100!'

IV.18 CONVPREP-Rules

```
(66,1)  CONVPREP <- /AS/         UNDEFINED
(66,2)  CONVPREP <- /TO/         UNDEFINED
(66,3)  CONVPREP <- /IN/         UNDEFINED
```

These are the prepositions used in the phrases giving the CONVUNIT. Certain prepositions are commonly used with certain verbs, for example, 'express as', 'change to', and 'write in', but they can be treated as equivalent here because the verbs are all changed to 'give' either by the TRANSL if that is their only use or the dictionary if there are other uses of the verb.

IV.19 SPECPREPHRASE-Rules

(61,1) SPECPREPHRASE <- SPECPREPHRASE /OR/ SPECPREPHRASE
(61,2) SPECPREPHRASE <- SPECPREPHRASE /AND/ SPECPREPHRASE
(61,3) SPECPREPHRASE <- V SPECPREP1
(61,4) SPECPREPHRASE <- /NOT/ V SPECPREP1
(61,5) SPECPREPHRASE <- /EITHER/ V SPECPREP1 /OR/ V SPECPREP1
(61,6) SPECPREPHRASE <- /NEITHER/ V SPECPREP1 /NOR/ V SPECPREP1
(61,7) SPECPREPHRASE <- SPECPREP1
(61,8) SPECPREPHRASE <- /NOT/ SPECPREP1
(61,9) SPECPREPHRASE <- /NEITHER/ SPECPREP1 /NOR/ SPECPREP1
(61,1[illegible]) SPECPREPHRASE <- /EITHER/ SPECPREP1 /OR/ SPECPREP1

(62,1) V <- =
(62,2) V <- /EXPRESSED/

These rules parse lists of the special prepositional phrases which are used to give CONVUNITS, and they also parse the complements of the phrases. Note that here as in several other places in the grammar the syntax for 'not' when it appears in various positions in a list has not been carefully worked out. There is a problem in deciding when an initial 'not' should be forced to extend to all members of the list and when it should not. The category V includes the verbs 'expressed' and 'equal'. These verbs may optionally precede the SPECPREP1. Examples are 'expressed as a fraction' and 'equal as a fraction'.

IV.20 SPECPREP1-Rules

(63,1) SPECPREP1 <- /INLOWESTTERMS/
(63,2) SPECPREP1 <- /INEXPANDEDFORM/
(63,3) SPECPREP1 <- CONVERSIONS

(64,1) CONVERSIONS <- CONVPREP CONVUNITS /OR/ CONVPREP CONVUNITS
(64,2) CONVERSIONS <- CONVPREP CONVUNITS
(64,3) CONVERSIONS <- CONVPREP CONVUNITS CC1
(64,4) CONVERSIONS <- CONVPREP CONVUNITS , CC1
(64,5) CONVERSIONS <- CONVPREP CONVUNITS CC1 CC1

(64,6) CONVERSIONS <- CONVPREP CONVUNITS , CC1 , CC1
(64,7) CONVERSIONS <- CONVPREP CONVUNITS , CC1 CC1
(64,8) CONVERSIONS <- CONVPREP CONVUNITS CC1 , CC1

(65,1) CC1 <- CONVPREP CONVUNITS
(65,2) CC1 <- /THEN/ CONVPREP CONVUNITS
(65,3) CC1 <- /AND/ CONVPREP CONVUNITS
(65,4) CC1 <- /AND/ /THEN/ CONVPREP CONVUNITS

'In lowest terms' and 'in expanded form' are very common phrases that I have TRANSL'd, but often the word 'form' appears with other CONVUNITs, for example, 'Is 2 in decimal form?' Another rule is needed.

I will not discuss the details of Rules (64,1)-(64,8) and (65,1)-(65,4). A wide variety of syntactic formats is used in these expressions and I have tried to write rules that will parse all the different formats. The following are examples of commands which will use these rules in their syntactic derivation:

a) Express 2 3/4 as a fraction!

b) Express 19 % as a fraction and then as a decimal!

c) Write 92 of 100 parts as a fraction, as a decimal, and as a percent!

d) Write 478 of 1000 parts as a fraction whose denominator is 100, 1000, or 10000, then as a decimal and a percent!

e) Convert 3.1 m. to decimeters, centimeters, and millimeters!

f) Find the sum of 1/5 and 3/5 and express the answer as a mixed number or a whole number, if possible!

IV.21 ORDERING-Rules

```
(69,1)  ORDERING <- /INORDER/
(69,2)  ORDERING <- /INORDER/ ORDERING1
(69,3)  ORDERING <- ORDERING1

(70,1)  ORDERING1 <- /STARTINGWITH/ EXP
(70,2)  ORDERING1 <- /STARTINGWITH/ DET COMPADJ
(70,3)  ORDERING1 <- /STARTINGWITH/ DET COMPADJ L/CNP3
(70,4)  ORDERING1 <- /FROM/ COMPADJ /TO/ COMPADJ
```

Certain phrases used to request an ordering of the answer have been TRANSL'd to 'inorder'. This phrase is essentially meaningless since the evaluator always orders the answer. The important element when present is the actual specification for the ordering. Rules (70,1) to (70,4) will parse several ways of giving these specifications. Examples are:

a) List the factors of 12 in order starting with the least factor!

b) List the factors of 12 in order from greatest to least!

c) Arrange 1 m., 1 cm., and 1 km. by size from smallest to largest.

IV.22 Commands Using Special Verbs

```
(60,1)  C <- /COUNT/ /TO/ EXP /BY/ LISTNAMESNU
(60,2)  C <- /COUNT/ /TO/ EXP /BY/ LISTNAMESNU /AND/ /TO/ EXP /BY/
                                                      LISTNAMESNU
(60,3)  C <- /COUNT/ /BY/ LISTNAMESNU /TO/ EXP
(60,4)  C <- /COUNT/ /BY/ LISTNAMESNU /TO/ EXP /AND/ /BY/ LISTNAMESNU
                                                          /TO/ EXP
(60,5)  C <- /REGROUP/ NUNITS /AS/ NUNITS
(60,6)  C <- /REGROUP/ NUNITS /AS/ NUNITS /AND/ /AS/ NUNITS
(60,7)  C <- /REGROUP/ NUNITS /AS/ NUNITS /AND/ NUNITS /AS/ NUNITS
(60,8)  C <- /SOLVE/ /THE/ /EQUATION/ F
(60,9)  C <- /SOLVE/ F
(60,10) C <- /ROUND/ EXP /TO/ UNITS
(60,11) C <- /ROUND/ EXP /TO/ UNITS /AND/ EXP /TO/ UNITS
(60,12) C <- /ROUND/ EXP /TO/ UNITS /AND/ /TO/  UNITS
```

Certain verbs used in commands need to be dealt with individually. I have included rules for 'count', 'regroup', 'solve', and 'round'. The semantic functions are currently UNDEFINED. Examples of these rules are:

1) Count to 100 by fives!
2) Count to 10 by ones and to 20 by twos!
3) Count by fives to 100!
4) Count by fives and tens to 100 and by hundreds to 1000!
5) Regroup 1 ten as 10 ones!
6) Regroup 1 hundred as 10 tens and 1 ten as 10 ones!
7) Regroup 1 hundred as 10 tens and 1 ten as 10 ones!
8) Solve the equations $b+c=6$ and $b-c=2$!
9) Solve $a+5=12$!
10) Round .6854 to tenths!
11) Round .853 to tenths and .9637 to hundredths!
12) Round .7596 to hundredths and to tenths!

The rules need to be consolidated so that separate rules are not needed for multiple specifications of arguments in the commands.

IV.23 Arithmetic Commands

```
(60,22)  C <- /ADD/ NP /TO/ NP           (ADDER ;2; ;4;)
(60,23)  C <- /SUBTRACT/ NP /FROM/ NP    (SUBBER ;4; ;2;)
(60,24)  C <- /MULTIPLY/ NP /BY/ NP      (MULTER ;2; ;4;)
(60,25)  C <- /DIVIDE/ NP /BY/ NP        (DIVVER ;2; ;4;)
(60,26)  C <- /DIVIDE/ NP /INTO/ NP      (DIVVER ;4; ;2;)
```

Often these expressions are found not as commands but embedded in other sentences. Examples are:

a) Find a fraction equal to 1/2 by multiplying the numerator and denominator by the same number!

b) To find an equal fraction, we can divide both the numerator and denominator by the same number.

These complex sentences have not been considered at this stage.

IV.24 Basic Command Rule

(60,21) C <- GV NP ;2;
(71,1) GV <- /GIVE/
(71,2) GV <- /GIVE/ PERSP

This is the most common form of command. Many verbs are included in the category /GIVE/. Rule (71,2) allows commands to be prefaced by 'give me'. Some illustrative examples are:

a) Give 2 pairs of even numbers whose sum is 12!
b) Find the set of whole numbers N such that N < 7.!
c) Find the members of the set {n : n < 5}!
d) Write the sum of 84, 57, and 76!
e) Name the numerator of 3/5!
f) Give 4 fractions equal to 1/4!
g) Give the set of the first ten multiples of 1!
h) List all the factors of 11!
i) Give the prime numbers between 65 and 80!

The evaluator simply outputs the result of evaluating the NP.

IV.25 Special Conversion Commands

(60,13) C <- GV CONVERSIONS NP
(60,14) C <- GV , CONVERSIONS , NP
(60,15) C <- GV CONVERSIONS , NP
(60,16) C <- GV NP ORDERING
(60,17) C <- /CONVERT/ NP /FROM/ CONVUNITS /TO/ CONVUNITS

Most of the commands involving requests for conversion are parsed by the basic command rule

(60,21) C <- GV NP
in combination with the following NP-rule:

(84,3) NP1 <- NP SPECPREPHRASE .
Some examples are:

a) Express 4/16 in lowest terms!
b) Write 2 3/4 as a fraction!
c) Convert 3' centigrade to Fahrenheit!

The semantics for rule (84,3) will be to convert the NP to the form named in the SPECPREPHRASE. So here again the evaluation is complete at the NP level.

However, in certain commands the order of the arguments is reversed so rules (60,13)-(60,15) have been written which allow optional use of commas. An example is:

d) Write in Roman numerals the number of months in a year!

Rule (60,17) parses conversion commands which include the type of the original expression as well as the type to be converted to. For example:

e) Convert 3/5 from a fraction to a decimal!

The semantic function will need to check that 3/5 is indeed a fraction before it converts to decimal.

Rule (60,16) parses commands which contain a request for the ordering of the answer. For examples see Section IV.21.

IV.26 Combinations of Commands

```
(60,18)  C <- C /AND/ /EXPRESSANSWER/ SPECPREP1
(60,19)  C <- C /AND/ /STATEIFEACH/ LINK SUBST
(60,20)  C <- C /AND/ C
```

Many commands include subcommands which are dependent on the main command, for example,

Write 3.7 + 2.1 in column form and find the sum!

We have not dealt with problems of reference either within a sentence or between sentences in this stage of the project and thus cannot handle most of these complex subcommands. However, rules (60,18) and (60,19) do handle two forms of subcommands. The first allows the questioner to state the desired form of the answer, for example,

Find the sum of 2/5 and 3/5 and express the answer
as a whole number!

The second allows a further question to be asked about the answer to the main command, for example,

Find all the factors of 15 and state whether each is prime!

Various phrases are TRANSL'd to /EXPRESSANSWER/ and /STATEIFEACH/.

Rule (60,20) will handle compound commands which are complete in themselves, for example 'Give all the even factors of 12 and give all the prime factors of 15!'

IV.27 Declaratives

I will not list any of the rules for declaratives. We have concentrated our efforts at this stage on questions and commands. There are two primary reasons for working with questions before declaratives. First, a study of [23] shows that there is less variety of syntax and vocabulary in questions than declaratives. Declaratives contain more 1) idiomatic expressions, 2) distinct uses of prepositions, 3) verbs, 4) phrases with no mathematical representation, 5) pronouns, and 6) references to other sentences in the text. Second, declaratives require a more sophisticated data base which can be added to or revised on the basis of the input and which can provide temporary storage for information. Temporary storage is required by some questions also (namely, those with embedded declaratives), but they have also been excluded in the first stage.

IV.28 NP-Rules

```
(83,1)   NP <- L/CNP1                                 ;1;
(83,2)   NP <- EXP                                    ;1;
(83,3)   NP <- /EITHER/ NP /OR/ NP                    (CHL ;2; ;4;)
(83,4)   NP <- /NEITHER/ NP /NOR/ NP                  (NOR ;2; ;4;)

(107,1)  L/CNP1 <- NP1CHOICE                          (CHL ;1;)
(107,2)  L/CNP1 <- LISTOFNP1                          (CHL ;1;)
(107,3)  L/CNP1 <- NP1                                ;1;

(102,1)  LISTOFNP1 <- NP1 /AND/ NP1                   ;1; ;3;
(102,2)  LISTOFNP1 <- LISTOFNP1 , NP1                 ;1; ;3;
(102,3)  LISTOFNP1 <- NP1 , NP1                       ;1; ;3;
(102,4)  LISTOFNP1 <- LISTOFNP1 , /AND/ NP1           ;1; ;4;
(102,5)  LISTOFNP1 <- LISTOFNP1 /AND/ NP1             ;1; ;3;

(39,1)   NP1CHOICE <- NP1 , NP1                       ;1; ;3;
(39,2)   NP1CHOICE <- NP1 /OR/ NP1                    ;1; ;3;
(39,3)   NP1CHOICE <- NP1CHOICE , NP1                 ;1; ;3;
(39,4)   NP1CHOICE <- NP1CHOICE , /OR/ NP1            ;1; ;4;
(39,5)   NP1CHOICE <- NP1CHOICE /OR/ NP1              ;1; ;3;
```

NP is the highest level category for noun phrases. The two basic types of noun phrases which can be derived from NP are EXP and NP1. The EXP's are standard arithmetic expressions and the NP1's are more flexible natural language noun phrases. At this time, EXP's and NP1's cannot be mixed in lists or choicelists. This needs to be changed since every EXP can in fact be written as an equivalent NP1. For example the EXP '1+2' evaluates to the same number as 'the sum of 1 and 2', 'the largest odd factor of 12', 'the positive square root of

9', etc. However, to allow mixed lists is also to allow ambiguities. Consider:

> EX1: Give the factors of 2, the factors of 3, and the factors of 4!

There will be two syntactic parses of EX1 if EXP's and NP1's can be mixed on lists. These parses reflect the following two interpretations of the sentence.

> EX1a: Give (1) the factors of 2, (2) the factors of the factors of 3, and (3) the factors of the factors of 4!

> EX1b: Give (1) the factors of 2, (2) the factors of 3 and (3) the factors of 4!

Clearly EX1b is the more plausible interpretation, but the ambiguity is genuine. Consider another example:

> EX2: Are the factors of 12, the largest factor of 12, and the smallest multiple of 12 equal?

> EX2a: Are the factors of 12 equal to the factors of the largest factor of 12 and also equal to the factors of the smallest multiple of 12?

> EX2b: Are the factors of 12 equal to the largest factor of 12 and also equal to the smallest multiple of 12?

Admittedly, this question is very unlikely to occur, but I chose it as an example because of its close parallel to EX1. It would be very difficult to distinguish 'the factors of 2, the factors of 3, and the factors of 4' from 'the factors of 12, the largest factor of 12, and the smallest multiple of 12' grammatically, but EX1b is the most plausible interpretation of EX1 while EX2a is the most plausible interpretation of EX2. The problem is not solved by prohibiting mixed lists because they are needed in many contexts and in this case if mixed lists were not allowed EX2 would be parsed only in the incorrect way. There does not appear to be any feasible way to rewrite the grammar to avoid the ambiguity. If two syntactic parses have the same semantic construction, there is no problem, but that is not the case here. Also, if only one semantic construction can be evaluated there is no problem, but both interpretations of EX1 as well as both interpretations of EX2 can be evaluated.

IV.29 NP1-Rules

```
(84,1)  NP1 <- L/CNP2 /EXCEPT/ NP                          (SD ;1; ;3;)
(84,2)  NP1 <- L/CNP2 RELPOSPRONS                          (I ;1; ;2;)
(84,3)  NP1 <- NP SPECPREPHRASE                            UNDEFINED
(84,4)  NP1 <- /THE/ EXP1 /IN/ EXP1                        UNDEFINED
(84,5)  NP1 <- DET N VAR /SUCHTHAT/ VAR ARITHRELS          UNDEFINED
(84,6)  NP1 <- DET N VAR /SUCHTHAT/ VAR ARITHRELS /AND/ VAR ARITHRELS
                                                           UNDEFINED
(84,7)  NP1 <- DET N VAR /SUCHTHAT/ VAR ARITHRELS /OR/ VAR ARITHRELS
                                                           UNDEFINED
(84,8)  NP1 <- DET N VAR /SUCHTHAT/ EXP2 ARITHREL VAR ARITHREL EXP2
                                                           UNDEFINED
(84,9)  NP1 <- /THE/ COMPADJ /OF/ NP                 (MAXF (FCN ;2;) ;4;)
(84,10) NP1 <- NP2                                         ;1;

(108,1) L/CNP2 <- NP2CHOICE                                (CHL ;1;)
(108,2) L/CNP2 <- LISTOFNP2                                (CHL ;1;)
```

Rules (84,1) and (84,2) parse expressions of exception and relative clauses which appear at the end of a list and are applied to each member of the list, for example

EX1: Give the factors of 12 and the factors of 15
that are prime! .

Every time these rules are used in a derivation, there will be another derivation of the NP in which the modifier is attached only to the last element of the list. These rules should actually be more sophisticated because the presence of a similar modifier on another element of the list rules out the interpretation in which the scope of the final modifier is extended. For example,

EX2: the factors of 4 that are even and the factors of 5 that are odd.

should not be parsed by Rule (84,2). The semantic function for rule (84,1) which parses explicit exceptions is SD (set difference). An example is:

EX3: All the factors of 12 except 3 are even.

The rest of the NP1 rules are for noun phrases which have a fairly rigid format and cannot be used in the formation of more complex noun phrases other than lists. They cannot for instance be modified by relative clauses. I will give one example of each.

(84,3) 5/10 in lowest terms
(84,4) the 9 in 893
(84,5) a number N such that $N < 5$
(84,6) a number N such that $N < 5$ and $N > 3$
(84,7) the numbers N such that $N < 5$ or $N > 10$
(84,8) a number N such that $2 < N < 4$
(84,9) the largest of 2 cm., 3 m., and 5 mm.

Rule (84,10) allows derivation of the more common noun phrases by the NP2-rules. The recursive rules for lists of the various levels of NP's have not been included here since they are the same as the LIST rules given in many of the preceding sections.

IV.30 NP2-Rules

```
(85,1)   NP2 <- /A/ L/CNP3                    ;2;
(85,2)   NP2 <- /THE/ L/CNP3                  ;2;
(85,3)   NP2 <- /THE/ /NUMBER/ L/CNP3         (ENMF ;2; ;3;)
(85,4)   NP2 <- /ALL/ QU/I/HMNP               ;2;
(85,5)   NP2 <- /ALL/ /NUMBER/ QU/I/HMNP      (ENMF ;2; ;3;)
(85,6)   NP2 <- /ALL/ /THE/ L/CNP3            ;3;
(85,7)   NP2 <- /ANY/ QU/I/HMNP               ;2;
(85,8)   NP2 <- /ANY/ /NUMBER/ QU/I/HMNP      (NUMF ;2; ;3;)
(85,9)   NP2 <- /SOME/ QU/I/HMNP              ;2;
(85,10)  NP2 <- /THE/ COMPADJ L/CNP3          (MAXF (FCN ;2;) ;3;)
(85,11)  NP2 <- /A/ COMPADJ L/CNP3 /THAN/ NP
                                              (I (APP (FCN ;2;) ;5;) ;3;)
(85,12)  NP2 <- /A/ COMPADJ L/CNP3 /THAN/ NP RELPOSPRONS
                                              (I (APP (FCN ;2;) ;5;)
                                                         (I ;3; ;6;))
(85,13)  NP2 <- /THE/ ORDADJ L/CNP3            (ORDFCN  1 ;2; ;3;)
(85,14)  NP2 <- /THE/ ORDADJ /NUMBER/ L/CNP3 (ORDFCN ;3; ;2; ;4;)
(85,15)  NP2 <- /THE/ ORDADJ COMPADJ L/CNP3  (ORDFCN 1 ;2;
                                                 (APP (FCN ;3;) ;4;))
(85,16)  NP2 <- /THE/ /NUMBER/ COMPADJ L/CNP3
                                          (ENMF ;2; (APP (FCN ;3;) ;4;))

(109,1)  L/CNP3 <- LISTOFNP3                   (CHL ;1;)
(109,2)  L/CNP3 <- NP3CHOICE                   (CHL ;1;)
(109,3)  L/CNP3 <- NP3                         ;1;
```

The determiners have been worked out for the HAVENP's but not for the regular NP's so I will not discuss them here. Because of the difficulties involved, it is reasonable to not tackle the problem of determiners at the start of the project but to wait until other aspects of the noun phrases are worked out and some experience has been gained.

Rules (85,10)-(85,16) deal with two of the special categories of adjectives, ORDADJ and COMPADJ, which are the ordinals and the comparatives. Note that L/CNP3 will derive a list, a choicelist, or a single NP3.

(85,10) the largest factor of 5
(85,11) a larger number than 2
(85,12) a larger factor of 4 than 2 that is even

(85,13) the first multiple of 10
(85,14) the first 2 multiples of 10
(85,15) the second largest factor of 12
(85,16) the 2 largest factors of 12

The category QU/I/HMNP is used with quantifiers and in interrogative and 'how many' questions. I will give each of the QU/I/HMNP rules followed by examples.

(48,1) QU/I/HMNP <- L/CNP3 ;1;

Which factors of 12 are even?
Are any factors of 30 even?

(48,2) QU/I/HMNP <- /OF/ /THE/ L/CNP3 ;3;

Which of the factors of 12 are even?
Are all of the factors of 12 even?

[Note: 'of the' adds no meaning.]

(48,3) QU/I/HMNP <- /OF/ /THE/ L/CNP3 EXP
(I ;3; ;4;)

Which of the fractions 1/2, 3/12, and 2/9 are in lowest terms?

[Note: This rule allows an explicit list to be given. In fact the rule is probably unneeded since it is ambiguous with the rule for appositive nouns, NP6 <- APPOSN EXP. Most N's also have the lexical category APPOSN.]

(48,4) QU/I/HMNP <- /OF/ /THE/ /NUMBER/ L/CNP3
(ENMF ;3; ;4;)

Are any of the 3 factors of 6 odd?

[Note: The semantic function ENMF checks to be sure that the NP3 does in fact have the cardinality given by /NUMBER/. In this case, it must match exactly or there is an error, for example 'the 5 factors of 6' and 'the 2 factors of 6' both contain an error.]

(48,5) QU/I/HMNP <- /OF/ /THE/ EXP1 L/CNP3 EXP
(ENMF ;3; (I ;4; ;5;))

Are any of the 3 fractions 1/2, 3/12, and
2/9 in lowest terms?

[Note: Again, this rule is unneeded because the case is already adequately handled by the appositive noun rule. Intersection is probably not the best semantic function for appositive nouns. Any elements on the list which were not fractions would simply be eliminated when in fact an error should be noted. The ENMF checks that the correct cardinality was given for the list.]

(48,6) QU/I/HMNP <- /OF/ /THE/ L/CNP3 /AND/ /THE/ /LCNP3
(CHL ;3; ;6;)

Which of the factors of 2 and the factors of
6 are prime?
Are all of the factors of 9 and the factors of
10 odd?

(48,7) QU/I/HMNP <- /OF/ /THE/ L/CNP3 /OR/ /THE L/CNP3
(CHL ;3; ;6;)

Are any of the factors of 9 or the
factors of 3 even?

(48,8) QU/I/HMNP <- QU/I/HMNP /EXCEPT/ NP
(SD ;1; ;3;)

Are all the factors of 12 except 1 and 3
even?

IV.31 NP3-Rules

(86,1) NP3 <- NP4 ;1;
(86,2) NP3 <- L/CNP3 RELPOSPRONS (I ;1; ;2;)

The adjectives and relative clauses both have the I semantic function so the order of derivation is unimportant. However, if the rules for both adjective and relative clause modifiers were at the same level of the grammar there would be ambiguous derivations since either could be parsed first. Therefore they are derived at different levels to avoid unnecessary ambiguity. The NP3-level derives the relative clauses. Note that rule (86,2) when applied recursively parses strings of relative clauses, for example, 'numbers that are less than 10 that are greater than 5'.

IV.32 NP4-Rules for Set Nouns

(87,1)	NP4 <- ADP L/CNP5	(I ;1; ;2;)
(87,2)	NP4 <- NP5	;1;
(101,1)	NP5 <- N	(STS ;1;)
(101,2)	NP5 <- N ARITHRELS	(I (STS ;1;) ;2;)
(101,3)	NP5 <- N PREPHRASE	(I (STS ;1;) ;2;)
(101,4)	NP5 <- NP6	;1;
(106,1)	NP6 <- PN	(STS ;1;)
(106,2)	NP6 <- /DAY/	(STS ;1;)
(106,3)	NP6 <- /WEEK/	(STS ;1;)
(106,4)	NP6 <- /DIMENSION/	(STS ;1;)
(106,5)	NP6 <- /3D/	(STS ;1;)
(106,6)	NP6 <- MONTH	(STS ;1;)
(106,7)	NP6 <- GEOFIGURE	(STS ;1;)
(106.8)	NP6 <- UNITLISTS GEOFIGURE	(I ;1; ;2;)
(106,9)	NP6 <- APPOSN EXP3	(I (STS ;1;) ;2;)
(106,10)	NP6 <- APPOSN , EXP3 ,	(I (STS ;1;) ;3;)
(106,11)	NP6 <- GEOFIGURE UNITLISTS	(I ;1; ;2;)
(106,12)	NP6 <- GEOFIGURE , UNITLISTS ,	(I ;1; ;3;)
(106,13)	NP6 <- GEOFIGURE RELPOSPRONS	(I ;1; ;2;)
(106,14)	NP6 <- /EQUATION/ F	;2;
(106,15)	NP6 <- /EQUATION/ , F ,	;3;

At the NP4-level the rules dealing with noun phrases containing FCN's can be separated from the rules dealing with noun phrases containing N's. I will discuss the rules for nouns representing sets (N's) in this section and the rules for function nouns (FCN's) in the next section.

As the relative clauses have a separate level (NP3), there is a separate level here for adjectives (NP4). The NP5 level will derive either an N alone or with an arithmetic relation or a PREPHRASE ('between', 'before', 'after', 'in'). Usually an arithmetic relation as a modifier is contained in a relative clause but not always.

EX1: The set of natural numbers less than 3 is {0,1,2} .

The NP6-rules are for the less common lexical categories of nouns. There are currently nine such categories. These categories need reworking and will not be discussed here.

IV.33 NP4-Rules for Function Nouns

```
(87,3)  NP4 <- ADJ/FCNL/C /OF/ NP        (APP ;1; ;3;)
(87,4)  NP4 <- FCNL/C /OF/ NP            (APP ;1; ;3;)
(87,5)  NP4 <- ADP AND/OR DET ADP FCN/REL /OF/ NP
                                         (CHL (I ;1; (APP ;5; ;7;))
                                              (I ;4; (APP ;5; ;7;)))
(87,6)  NP4 <- ADP FCNL/C /OF/ NP        (I ;1; (APP ;2; ;4;))

(96,1)  FCNL/C <- FCNLIST                ;1;
(96,2)  FCNL/C <- FCNCHOICE              ,1;
(96,3)  FCNL/C <- FCN/REL                ;1;

(100,1) FCN/REL <- FCN                   (FCN ;1;)
(100,2) FCN/REL <- FCN RELPOSPRONS       (FCNMK (FCN ;1;) ;2;)

(97,1)  FCNLIST <- FCN/REL /AND/ FCN1               (CHL ;1; ;3;)
(97,2)  FCNLIST <- FCN/REL , FCN1 , /AND/ FCN1      (CHL ;1; ;3; ;6;)
(97,3)  FCNLIST <- FCN/REL , FCN1 /AND/ FCN1        (CHL ;1; ;3; ;5;)

(98,1)  FCNCHOICE <- FCN/REL /OR/ FCN1              (CHL ;1; ;3;)
(98,2)  FCNCHOICE <- FCN/REL , FCN1 , /OR/ FCN1     (CHL ;1; ;3; ;6;)
(98,3)  FCNCHOICE <- FCN/REL , FCN1 /OR/ FCN1       (CHL ;1; ;3; ;5;)

(99,1)  FCN1 <- FCN/REL                  ;1;
(99,2)  FCN1 <- DET FCN/REL              ;2;

(118,1) DET <- /A/                       ;1;
(118,2) DET <- /THE/                     ;1;
```

```
(95,1)  ADJ/FCNL/C <- ADJ/FCNLIST        ;1;
(95,2)  ADJ/FCNL/C <- ADJ/FCNCHOICE      ;1;

(92,1)  ADJ/FCNCHOICE <- ADP FCN/REL /OR/ ADJFCN1
                                         (CHL (FCNMK ;2; ;1;) ;4;)
(92,2)  ADJ/FCNCHOICE <- ADP FCN/REL , ADJFCN1 , /OR/ ADJFCN1
                                         (CHL (FCNMK ;2; ;1;) ;4; ;7;)
(92,3)  ADJ/FCNCHOICE <- ADP FCN/REL , ADJFCN1 /OR/ ADJFCN1
                                         (CHL (FCNMK ;2; ;1;) ;4; ;6;)

(93,1)  ADJ/FCNLIST <- ADP FCN/REL /AND/ ADJFCN1
                                         (CHL (FCNMK ;2; ;1;) ;4;)
(93,2)  ADJ/FCNLIST <- ADP FCN/REL , ADJFCN1 , /AND/ ADJFCN1
                                         (CHL (FCNMK ;2; ;1;) ;4; ;7;)
(93,3)  ADJ/FCNLIST <- ADP FCN/REL , ADJFCN1 /AND/ ADJFCN1
                                         (CHL (FCNMK ;2; ;1;) ;4; ;7;)

(94,1)  ADJFCN1 <- ADP FCN/REL           (FCNMK ;2; ;1;)
(94,2)  ADJFCN1 <- DET ADP FCN/REL       (FCNMK ;3; ;2;)
```

There is a problem with determining the scope of determiners and modifiers in noun phrases. Until this problem has been solved, I have assumed in the above rules that lists of function names will either have a single adjective at the head of the list which is intended to cover each element of the list or an adjective attached to each function name on the list (or, of course, no adjectives at all). Rule (87,4) is for the case with no adjectives, Rule (87,6) for a single adjective with a single function name or a list of function names, and Rule (87,3) for lists or choices (not singleton lists) where each element has an adjective. Another problem is that lists will at this level already have had the initial determiner (if any) parsed by an NP2 rule. There may or may not be determiners attached to the other elements of the list. Note that it is very unlikely that these determiners will differ from the initial determiner so they can be ignored.

EX1: the sum and the product of 3 and 4
EX2: the even factors and prime factors of 12

Lists whose elements have different determiners will usually have each element fully specified, for example

EX3: all the factors of 2 and 3 factors of 12

and are therefore parsed as LISTOFNP1's rather than by these rules. The category FCN1 has been added to parse the optional determiner preceding elements other than the first of the list of functions.

Rule (87,5) parses noun phrases like

EX4: the even or the odd factors of 12 .

The semantic function needs to make two calls on the FACTOR function with one of the adjectives associated with each call and to designate 12 as the argument to each call on the FACTOR function.

The category FCN/REL allows for an optional relative clause following the function name, for example,

EX5: the factors that are even of 12 .

The semantic function FCNMK has been used in rule (100,2). This semantic function was discussed in Section II.3 on the HAVENP's. It creates new functions. In the above example, it will create an EVENFACTOR function. In order to use the intersection function for this example, the function, the relative clause, and the argument to the function would all need to be parsed at the same level. If this were done the semantics could apply the function to the argument and then intersect the result with the relative clause. In fact, the function and the argument are parsed by rule (87,4) where the semantic function is APP (apply). However, to also parse the relative clause at this level, would mean that recursive rules for lists could not be used. Each case would need a separate rule at the NP4 level.

Rules (87,3) and rules (95,1)-(95,2), (92,1)-(92,3), (93,1)-(93,3), and (94,1)-(94,2) are used for lists of functions each of which has an associated adjective. Here again FCNMK is used because the adjective, function name, and argument cannot all be parsed at the NP4-level unless the use of recursive rules for lists is sacrificed.

IV.34 2FCN-Rules

```
(87,7)  NP4 <- ADP 2FCNL/C /OF/ LSTNP/EXP     (I ;1; (APP ;2; ;4;))
(87,8)  NP4 <- ADP AND/OR DET ADP 2FCN /OF/ LSTNP/EXP
                                              (CHL (I ;1; (APP ;5; ;7;))
                                                   (I ;4; (APP ;5; ;7;)))
(87,9)  NP4 <- 2FCNL/C /OF/ LSTNP/EXP         (APP ;1; ;3;)

(88,1)  LSTNP/EXP <- LISTOFNP1                (LST ;1;)
(88,2)  LSTNP/EXP <- LISTOFEXP                (LST ;1;)

(89,1)  2FCNL/C <- 2FCN AND/OR 2FCN1             (CHL (FCN ;1;) ;3;)
(89,2)  2FCNL/C <- 2FCN , 2FCN1 , AND/OR 2FCN1   (CHL (FCN ;1;) ;3; ;6;)
(89,3)  2FCNL/C <- 2FCN , 2FCN1 AND/OR 2FCN1     (CHL (FCN ;1;) ;3; ;5;)
(89,4)  2FCNL/C <- 2FCN                          (FCN ;1;)

(90,1)  2FCN1 <- 2FCN                         (FCN ;1;)
(90,2)  2FCN1 <- DET 2FCN                     (FCN ;2;)

(91,1)  AND/OR <- /AND/
(91,2)  AND/OR <- /OR/
```

The category 2FCN is for function nouns which always have two or more arguments, for example, intersection, sum, and common factors. The category LSTNP/EXP is used for the argument to the 2FCN. The semantic function for these argument lists in Rules (88,1) and (88,2) is LST. Currently, the rules for parsing lists and choicelists of 2FCN's (Rules (89,1)-(89,4)) have been combined using the category AND/OR. No examples of relative clauses embedded in 2FCN noun phrases have been found so the FCN/REL option has not been included for them. Because adjectives are so infrequently used in this context I have in these rules allowed only a single adjective preceding a list of 2FCN's. A better approach might be to divide the category 2FCN into two separate categories for the 2FCN's like 'intersection' which never have adjective modifiers and the 2FCN's like 'common factor' which can be modified by adjectives.

IV.35 Existence Questions

```
(43,1)  Q <- LINK /THERE/ L/CNP1        (EXIST ;3;)
(43,2)  Q <- LINK /THERE/ L/CNP3        (EXIST ;3;)
```

Rules (43,1) and (43,2) are for questions of the form 'Are there ...?' The need for two rules is again caused by the incompletion of the rules for determiners. The semantic function EXIST checks whether or not the set is empty.

IV.36 If Questions

```
(43,13)  Q <- /IF/ D , Q          (IFTHEN (LST ;2; ;4;))
(43,14)  Q <- /IF/ F , Q          (IFTHEN (LST ;2; ;4;))
```

There has not yet been any implementation of the evaluation of 'if-then' questions. The 'if' part may be either a declarative or an arithmetic formula.

IV.37 Idiomatic Question Formats

```
(43,28)  Q <- /DOYOUKNOW/ NP                        ;2;
(43,29)  Q <- /DOYOUKNOW/ INTER NP LINK             ;3;
(43,31)  Q <- /DOYOUKNOW/ INTER EXP1 HAVENP NP /HAVE/
                                                    (NMF ;3; (APP ;4; ;5;))
(43,32)  Q <- /CANYOU/ C                            ;2;
```

These rules show one way that idiomatic expressions can be handled. In this approach, the common expressions need to be identified and either TRANSL'd to an already existing expression which plays the same role in the grammar or, if the grammatical role of the new expression is unique, new rules written. Thus 'are you familiar with' could be TRANSL'd to 'do you know' and many expressions could be TRANSL'D to 'give' as it is used in commands. There are already many synonyms for 'give' in the TRANSL and many more possibilities. It is a serious question as to whether this is the right approach. We need to know first how many such expressions there will be in actual use and also how many common grammatical constructions there are that are not covered by the present grammar. Unless these numbers prove to be very small which is unlikely, the method of manual addition to the TRANSL and the grammar is not feasible. Instead some other approach to the habitability problem will be needed. Examples of the above rules are:

(43,28) Do you know the largest common factor of 6 and 15?
(43,29) Do you know what the sum of 5 and 12 is?
(43,30) Do you know what even factors 12 has?
(43,31) Do you know which 6 factors 12 has?
(43,32) Can you give the factors of 12?

IV.38 Questions With Introductory Clauses

```
(43,36)  Q <- /EXCEPT/ NP , Q     (SD ;4; ;2;)
```

There are undoubtedly many other introductory clauses which can precede questions, but we have at this stage included only the rule for a clause stating an exception, for example,

EX1: Except for the number 2, are there any even prime numbers? Prepositional phrases using 'in' are common introductory phrases, for example,

EX2: In the fraction 2/3, which number is the denominator? Most of these other introductory clauses require intra-sentence referencing and more sophisticated use of the data base than is currently implemented.

IV.39 Questions Beginning with a Linking Verb

```
(43,10) Q <- LINK NP SUBST/EXP        (S ;2; ;3;)
(43,11) Q <- LINK /NUMBER/ L/CNP3 SUBST/EXP
                                      (S (LST (CARDINALITY (I ;3; ;4;)))
                                         (APP (FCN @EQL) (LST ;2;)))
(43,12) Q <- LINK CHOICELIST COMPADJ  (MAXF (FCN ;3;) ;2;)
(41,1)  CHOICELIST <- EXPCHOICE       (CHL ;1;)
(41,2)  CHOICELIST <- NP1CHOICE       (CHL ;1;)
```

Questions parsed by rule (43,10) were the most commonly found questions in [23]. Examples are:

Is 2 even?
Is 10 a multiple of 5?
Is 4 < 5?
Is 6 between 1 and 10?
Is 5 a multiple of 10 or a factor of 10?
Is 2 or 3 odd?

The semantic function for this rule is subset. More rules are needed for this question form which are sensitive to the determiner used. For example, EX1: Are any factors of 9 even?

which has an existential quantifier should use intersection rather than subset. An example of rule (43,11) is:

EX2: Are 2 factors of 12 odd?

I have not included the rule

RULE1: NP2 <- /NUMBER/ NP3

in the NP rules, but have instead included rules such as this one at the top level. In line with the objective of shifting a large part of the workload from the grammar to the semantic functions, RULE1 should be implemented with a suitable semantic function. The rule

```
RULE2:  NP2 <- /THE/ /NUMBER/ NP3        (ENMF ;2; ;3;)
```

is currently included in the NP-rules. The semantic function ENMF checks that the cardinality of NP3 matches the /NUMBER/ exactly.

EX3: Are the 3 factors of 9 odd?

An example of rule (43,12) is

EX4: Is 2 or 4 larger?

The only type of NP used with a COMPADJ is a CHOICELIST.

IV.40 Questions Beginning with an Auxiliary Verb

```
(43,4)   Q <- AUXIL NP /COME/ /BEFORE/ NP    (S ;2; (BEFORE ;5;))
(43,5)   Q <- AUXIL NP /COME/ /AFTER/ NP     (S ;2; (AFTER ;5;))
(43,9)   Q <- AUX NP ARITHCHOICE NP  (PICKFCN (CHL ;3;) (LST ;2; ;4;))
(43,3)   Q <- AUXIL1 NP VEQUAL NP    (S ;2; (APP (FCN EQL) ;4;))
(43,33)  Q <- MOD NP /BE/ SUBST/EXP  (S ;2; ;4;)
(43,34)  Q <- MOD  /NUMBER/ L/CNP3 /BE/ SUBST/EXP
                                     (S (LST (CARDINALITY (I ;3; ;5;)))
                                        (APP (FCN @EQL) (LST ;2;)))
(43,35)  Q <- MOD /NUMBER/ /BE/ ARITHCHOICE EXP1
                                     (PICKFCN (CHL ;4;) (LST ;2; ;5;))
(51,1)   AUXIL <- AUX
(51,2)   AUXIL <- MOD
```

```
(42,1)    AUXIL1 <- /DOES/
(42,2)    AUXIL1 <- MOD

(79,1)    VEQUAL <- =
(79,2)    VEQUAL <- /NAME/

(117,1)   ARITHCHOICE <- ARITHREL /OR/ ARITHREL         (FCN ;1;) (FCN ;3;)
(117,2)   ARITHCHOICE <- ARITHCHOICE /OR/ ARITHREL     ;1; (FCN ;3;)
(117,3)   ARITHCHOICE <- ARITHREL , ARITHREL            (FCN ;1;) (FCN ;3;)
(117,4)   ARITHCHOICE <- ARITHCHOICE , /OR/ ARITHREL ;1; (FCN ;4;)
```

Examples of rules (43,4) and (43,5) are:

Does 6 come before 7?
Will the product of 2 and 4 come before the sum of 2 and 4?

AUXIL can be either an auxiliary or a modal verb. ARITHCHOICE is a choicelist of arithmetic relations. Questions parsed by rule (43,9) are common in elementary textbooks. Examples are:

Does 2 = or not = 4/2?
Is .6 =, <, or > 60%?

Questions like 'Is 2 equal to 4/2?' are parsed as LINK NP ARITHREL NP because 'equal to' is TRANSL'd to '='. Rule (43,3) will parse 'Does 2 equal 4/2?' To avoid ambiguity, this rule uses the category AUXIL1 which includes 'does' and the modal verbs like 'can' and 'will' but excludes the verb 'to be' which is a linking as well as an auxiliary verb. The category VEQUAL in this rule includes the verbs 'equal' and 'name' and may easily be extended if any other verbs are found to have the semantics of equal in this context. As we extend the vocabulary, rules will be needed to parse verbs with different meanings in this position, for example, 'Does 7 factor 14?'

Rules (43,33), (43,34) and (43,35) use modal verbs with the verb 'to be'. One example of each follows:

(43,33) Will the sum of 2 and 3 be odd or even?
(43,34) Will 3 common multiples of 2 and 3 be less than 20?
(43,35) Will the product of 2 and 5 be <, >, or = to the sum of 2 and 5?

IV.41 CHOICELIST Questions

```
(43,15)  Q <- INTER2 LINK ADJ PUNCHOICE CHOICELIST
                              (S ;5; (STS ;3;))
(43,16)  Q <- INTER2 LINK COMPADJ PUNCHOICE CHOICELIST
                              (PICK (FCN ;3;) ;5;)
(43,17)  Q <- INTER2 LINK /THE/ COMPADJ PUNCHOICE CHOICELIST
                              (PICK (FCN ;4;) ;6;)
(43,18)  Q <- INTER2 LINK /THE/ ORDADJ COMPADJ PUNCHOICE CHOICELIST
                              (ORDFCN ;4; 1 (APP (FCN ;5;) ;7;)
(43,19)  Q <- INTER2 LINK DET COMPADJ N PUNCHOICE CHOICELIST
                              (PICK (FCN ;4;) (I (STS ;5;) ;7;))
(43,20)  Q <- INTER2 LINK DET COMPADJ ADP N PUNCHOICE CHOICELIST
                              (PICK (FCN ;4;) (I ;5; (I (STS ;6;) ;8;)))
(43,21)  Q <- INTER2 MOD /BE/ SUBST/EXP PUNCHOICE CHOICELIST
                              (I ;4; ;6;)

(43,25)  Q <- INTER2 LINK ARITHRELS PUNCHOICE CHOICELIST
                              (PICK ;3; ;5;)
(43,26)  Q <- INTER2 LINK PREPHRASE PUNCHOICE CHOICELIST
                              (PICK ;3; ;5;)
(43,27)  Q <- INTER2 LINK SPECPREPHRASE PUNCHOICE CHOICELIST
                              (PICK ;3; ;5;)

(47,1)  INTER2 <- INTER
(47,2)  INTER2 <- INTER /OF/ /THESE/
(47,3)  INTER2 <- INTER /OF/ /THESE/ L/CNP3

(49,1)  PUNCHOICE <- ,
(49,2)  PUNCHOICE <- :
(49,3)  PUNCHOICE <- ;
(49,4)  PUNCHOICE <- DASHES

(50,1)  DASHES <- -
(50,2)  DASHES <- DASHES -
```

Many questions in elementary textbooks use a multiple choice format. In this section I will discuss the questions which contain the answer choicelist as an integral part of the question. The category INTER2 can have three forms as shown in the following examples:

(47,1) Which is less than 5 -- 4 or 6?
(47,2) Which of these is even: 2,3, or 4?
(47,3) Which of these numbers is even; 2,3, or 4?

I am treating all these forms as semantically equivalent. In order to typecheck the NP in the third form with each of the answers, twice as many rules would be needed in this portion of the grammar. Since an error here seems to be unlikely, I have written the rules so that the NP when present is simply ignored. The category PUNCHOICE allows for a variety of punctuation.

(43,15) Which of these is even -- 2, 3, or 4?
(43,16) Which is larger, 2 or 5/2?
(43,17) Which is the largest: 5/2, 5/3, or 5/4?
(43,18) Which is the second largest: 5/2, 5/3, or 5/4?
(43,19) Which is the largest number: 2, 5, or 7?
(43,20) Which is the largest even number: 2, 5, or 7?
(43,21) Which of these will be even: the sum of 5 and 2, the difference of 5 and 2, or the product of 5 and 2?
(43,25) Which of these is divisible by 3: 2, 4, 6, or 8?
(43,26) Which of these numbers is between 5/2 and 5 -- 2, 4, or 6?
(43,27) Which of these is in lowest terms: 10/17, 10/15, or 10/12?

The choicelists in the Q-rules at the beginning of this section are an integral part of the question; without the choicelist, the question makes no sense. For example, one would not ask 'Which is even?' without giving a choice of possible answers. Similarly, 'Which is the largest number?' and 'Which is the largest even number?' do not make sense without a choicelist of answers. Note that these rules use the category N. An FCN would never be used in this position (unless an argument to the function were added to the rule). For example, 'Which is the largest factor, 2 or 8?' and 'Which is the smallest denominator, 1/2 or 2/3?' are not legitimate questions. (The same questions with 'have' instead of 'to be' are legitimate and will be discussed in Section II.3.) Questions parsed by the Q1-rules rather than the Q-rules can be asked alone or followed by a choicelist. When there is a choicelist, the question is evaluated independently of the choicelist and then the answer is compared with the choices. An example parsed by a Q1-rule is 'How many even numbers are prime -- 1, 2, or 3?'

IV.42 Q1-Rules

```
(43,37)  Q <- Q1                              ;1;
(43,38)  Q <- Q1 PUNCHOICE CHOICELIST         (PICK ;1; ;3;)
```

These rules allow the optional choicelist of answers for certain questions. I have not included rules for the ordinary multiple-choice question format where the answers are enumerated on separate lines following the question using a letter or number to identify each choice, but the same semantic functions can handle the ordinary multiple-choice format.

IV.43 HOWMANY Questions Involving UNITs and NUNITs

```
(44,26)  Q1 <- /HOWMANY/ LISTNAMESU AUXIL UNITS ALLV1
                                   (CONVERT (UNT ;2;) ;4;)
(44,27)  Q1 <- /THERE/ LINK /HOWMANY/ LISTNAMESU /IN/ UNITS
                                   (CONVERT (UNT ;4;) ;6;)
(44,28)  Q1 <- /HOWMANY/ LISTNAMESU INSIDE UNITS
                                   (CONVERT (UNT ;2;) ;4;)
(44,29)  Q1 <- /THERE/ LINK /HOWMANY/ LISTNAMESNU /IN/ EXP2
                                   (CONVERTNUM (UNT ;4;) ;6;)
```

```
(44,30)  Q1 <- /HOWMANY/ LISTNAMESNU INSIDE EXP2
                                  (CONVERTNUM (UNT ;2;) ;4;)
(44,31)  Q1 <- /HOWMANY/ LISTNAMESNU AUXIL EXP2 ALLV1
                                  (CONVERTNUM (UNT ;2;) ;4;)

(45,1)  INSIDE <- LINK /THERE/ /IN/
(45,2)  INSIDE <- /IN/
(45,3)  INSIDE <- LINK /IN/
(45,4)  INSIDE <- LINK
(45,5)  INSIDE <- AUX =
(45,6)  INSIDE <- =
(45,7)  INSIDE <- /SHOWNBY/

(77,1)  ALLV <- =
(77,2)  ALLV <- /NAME/
(77,3)  ALLV <- /SHOW/
(77,4)  ALLV <- /BE/

(78,1)  ALLV1 <- ALLV
(78,2)  ALLV1 <- /HAVE/
(78,3)  ALLV1 <- /GIVE/
```

LISTNAMESU and LISTNAMESNU are lists (including the singleton list) of the names of UNIT's and NUNIT's. For example,

There are how many yards, feet and inches in 125 inches?
There are how many ones, tens, and hundreds in 594?

The category ALLV1 used in rules (44,26) and (44,31) includes several verbs, but the question has the same meaning whichever verb is used.

How many feet does 24 inches have (equal)?
How many tens does 236 have (show, give, name)?

The category INSIDE in rules (44,28) and (44,30) also parses several constructions which have the same meaning in these contexts.

How many tens are there in 87?
How many inches in a foot?
How many tens are in 100?
How many feet is 36 inches?
How many pounds are equal to 2 tons?
How many teaspoons equal 1 tablespoon?
How many tens are shown by 850?

The semantic functions CONVERTNUM and CONVERT used for this type of HOWMANY question convert the UNIT or NUNIT to the form or forms specified by the LISTNAMESU or LISTNAMESNU.

IV.44 Other HOWMANY Questions

```
(44,32)  Q1 <- /THERE/ LINK /HOWMANY/ L/CNP3
                                  (LST (CARDINALITY ;4;))
(44,33)  Q1 <- /HOWMANY/ L/CNP3 LINK /THERE/
                                  (LST (CARDINALITY ;2;))
(44,34)  Q1 <- /HOWMANY/ QU/I/HMNP LINK /THERE/ RELPOSPRONS
                                  (LST (CARDINALITY (I ;2; ;5;)))
(44,35)  Q1 <- /HOWMANY/ L/CNP3 LINK /THERE/ /OF/ NP
                                  (LST (CARDINALITY (I ;2; ;6;)))
(44,36)  Q1 <- /HOWMANY/ QU/I/HMNP LINK /THERE/ PREPHRASE
                                  (LST (CARDINALITY (I ;2; ;5;)))
(44,37)  Q1 <- /HOWMANY/ QU/I/HMNP LINK /THERE/ SPECPREPHRASE
                                  UNDEFINED
(44,38)  Q1 <- /HOWMANY/ L/CNP3  (LST (CARDINALITY ;2;))
(44,39)  Q1 <- /HOWMANY/ QU/I/HMNP AUXIL NP =
                                  UNDEFINED
(44,44)  Q1 <- /HOWMANY/ QU/I/HMNP = NP
                        (LST (CARDINALITY (I ;2; (APP (FCN EQL) ;4;))))
(44,45)  Q1 <- /HOWMANY/ QU/I/HMNP LINK SUBST/EXP
                                  (LST (CARDINALITY (I ;2; ;4;)))
```

The category QU/I/HMNP was discussed in Section IV.30. The semantic function for these HOWMANY questions is CARDINALITY. The following are examples:

(44,32) There are how many even factors of 12?
(44,33) How many even primes are there?
(44,34) How many factors of 12 are there that are divisible by 3?
(44,35) How many factors are there of 2?
(44,36) How many prime numbers are there between 10 and 20?
(44,37) How many members of the set {1/2, 2/4, 3/8, 4/8} are there in lowest terms?
(44,38) How many factors of 12?
(44,39) How many members of the set {2/4, 3/6, 5/8} does 1/2 equal?
(44,44) How many members of the set {2/4, 3/6, 5/8} equal 1/2?
(44,45) How many numbers between 5 and 10 are prime?

IV.45 Interrogative Questions

The following interrogative questions as well as all the HOWMANY questions in the preceding sections use Q1-rules and therefore can be followed by a choicelist of answers using rule (43,38).

I will list and give an example of each of the interrogative question rules.

```
(44,1)  Q1 <- INTER2 AUXIL NP ALLV                    ;3;
            What does 3+5 equal?

(44,2)  Q1 <- INTER QU/I/HMNP AUXIL NP ALLV          (I ;2; ;4;)
            Which whole number does 9/3 equal?

(44,3)  Q1 <- INTER2 AUXIL1 NP SPECPREPHRASE         (I ;3; ;4;)
            What does 60% equal as a fraction?

(44,4)  Q1 <- INTER QU/I/HMNP AUXIL1 NP SPECPREPHRASE    UNDEFINED
            What time does 1300 equal in 12-hour time?

(44,8)  Q1 <- INTER2 LINK NP                          ;3;
            What is the sum of 5 and 8?

(44,10)  Q1 <- INTER QU/I/HMNP MOD /BE/ SUBST/EXP  (I ;2; ;5;)
             Which factors of 12 will be prime?

(44,11)  Q1 <- INTER QU/I/HMNP LINK SUBST/EXP       (I ;2; ;4;)
             Which of the fractions 1/2, 5/4, and 6/8 are proper
                  fractions?

(44,12)  Q1 <- INTER QU/I/HMNP LINK /THE/ COMPADJ  (MAXF (FCN ;5;) ;2;)
             Which member of the set {2/3, 5/6, 7/10} is the largest?

(44,13)  Q1 <- INTER QU/I/HMNP LINK /THE/ ORDADJ COMPADJ  UNDEFINED
             Which member of the set {2/3, 5/6, 7/10}
                            is the second largest?

(44,14)  Q1 <- INTER QU/I/HMNP LINK COMPADJ         (MAXF (FCN ;4;) ;2;)
             Which prime number is smallest?

(44,18)  Q1 <- INTER EXP1 QU/I/HMNP AUXIL NP ALLV  UNDEFINED
             Which 2 members of the set {2/4, 3/5, 4/8, 5/9}
                  does 1/2 equal?

(44,21)  Q1 <- INTER EXP1 QU/I/HMNP = NP            UNDEFINED
```

```
            Which 2 members of the set [2/4, 3/5, 4/8, 5/9]
                equal 1/2?

(44,22)  Q1 <- INTER EXP1 QU/I/HMNP MOD /BE/ SUBST/EXP
                                   (NMF ;2; (I ;3; ;6;))
            Which 2 factors of 12 will be prime?

(44,23)  Q1 <- INTER EXP1 QU/I/HMNP LINK SUBST/EXP
                                   (NMF ;2; (I ;3; ;5;))
            Which 4 numbers between 10 and 20 are prime?
```

IV.46 FCNHNP-Rules

```
(44,5)   Q1 <- INTER FCNHNP AUXIL NP /HAVE/
                                   (APP ;2; ;4;)
(44,6)   Q1 <- INTER EXP1 FCNHNP AUXIL NP /HAVE/
                                   (NMF ;2; (APP ;3; ;5;))
(44,41)  Q1 <- /HOWMANY/ FCNHNP AUXIL NP /HAVE/
                                   (LST (CARDINALITY (APP ;2; ;4;)))
(123,1)  FCNHNP <- FCN             (FCN ;1;)
(123,2)  FCNHNP <- ADP FCN         (FCNMK (FCN ;2;) ;1;)
(123,3)  FCNHNP <- FCN RESTRICT    (FCNMK (FCN ;1;) ;2;)
(123,4)  FCNHNP <- ADP FCN RESTRICT
                                   (FCNMK (FCN ;2;) (I ;1; ;3;))

(119,1)  RESTRICT <- RESTRICT RESTRICT1         (I ;1; ;2;)
(119,2)  RESTRICT <- RESTRICT1                  ;1;

(120,1)  RESTRICT1 <- PREPHRASE                 ;1;
(120,2)  RESTRICT1 <- RELPOSPRONS               ;1;
(120,3)  RESTRICT1 <- ARITHRELS                 ;1;
```

The noun phrase categories created for use with the verb 'to have' were discussed in detail in Section II.3. This section and the following sections give the rules for the various HNP categories and the Q-rules which use the categories.

IV.47 HNPAS-Rules

```
(44,7)  Q1 <- INTER QU/I/HMNP AUXIL HNPAS
                                  (I ;2; ;4;)
(44,20) Q1 <- INTER EXP1 QU/I/HMNP AUXIL HNPAS
```

```
                                  (NMF ;2; (I ;3; ;5;))
(44,40) Q1 <- /HOWMANY/ QU/I/HMNP AUXIL HNPAS
                                  (LST (CARDINALITY (I ;2; ;4;)))
(128,1) HNPAS <- NP /HAVE/ /AS/ ANSGVHNP
                                  (APP ;4; ;1;)
(128,2) HNPAS <- NP /HAVE/ /AS/ ANSGVHNP /OR/ /AS/ ANSGVHNP
                                  (CHL (APP ;4; ;1;) (APP ;7; ;1;))
(128,3) HNPAS <- NP /HAVE/ /AS/ ANSGVHNP /AND/ /AS/ ANSGVHNP
                                  (I (APP ;4; ;1;) (APP ;7; ;1;))

(130,1) ANSGVHNP <- ANSGV1HNP           ;1;
(130,2) ANSGVHNP <- DET ANSGV1HNP      ;2;

(131,1) ANSGV1HNP <- FCNHNP                              ;1;
(131,2) ANSGV1HNP <- /EITHER/ ANSGVHNP /OR/ ANSGVHNP    (FCNU ;2; ;4;)
(131,3) ANSGV1HNP <- /NEITHER/ ANSGVHNP /NOR/ ANSGVHNP (FCNNOR ;2; ;4;)
(131,4) ANSGV1HNP <- ANSGVHNP /AND/ ANSGVHNP            (FCNI ;1; ;3;)
```

IV.48 COMP1HNP and COMP2HNP-Rules

```
(43,8)   Q <- AUXIL NP /HAVE/ COMP2HNP
(43,24)  Q <- INTER2 /HAVE/ COMP1HNP PUNCHOICE CHOICELIST
(44,17)  Q1 <- INTER QU/I/HMNP /HAVE/ COMP1HNP
(44,25)  Q1 <- INTER EXP  QU/I/HMNP /HAVE/ COMP1HNP
(44,43)  Q1 <- /HOWMANY/ QU/I/HMNP /HAVE/ COMP1HNP
(132,1)  COMP2HNP <- COMPADJ /OR/ COMPADJ FCNHNP /THAN/ NP
(132,2)  COMP2HNP <- COMPADJ /OR/ COMPADJ FCNHNP /THAN/ NP /HAVE/
(132,3)  COMP2HNP <- ARITHREL /OR/ ARITHREL EXP FCNHNP
(132,4)  COMP2HNP <- COMPADJ FCNHNP /OR/ COMPADJ FCNHNP /THAN/ NP
(132,5)  COMP2HNP <- COMPADJ FCNHNP /OR/ COMPADJ FCNHNP /THAN/ NP
                                                                 /HAVE/
(132,6)  COMP2HNP <- DET COMPADJ /OR/ COMPADJ FCNHNP /THAN/ NP
(132,7)  COMP2HNP <- DET COMPADJ /OR/ COMPADJ FCNHNP /THAN/ NP /HAVE/
(132,8)  COMP2HNP <- DET COMPADJ /OR/ DET COMPADJ FCNHNP /THAN/ NP
(132,9)  COMP2HNP <- DET COMPADJ /OR/ DET COMPADJ FCNHNP /THAN/ NP
(132,10) COMP2HNP <- DET COMPADJ FCNHNP /OR/ DET COMPADJ FCNHNP /THAN/
                                                                      NP
(132,11) COMP2HNP <- DET COMPADJ FCNHNP /OR/ DET COMPADJ FCNHNP /THAN/
                                                               NP /HAVE/
(132,12) COMP2HNP <- DET COMPADJ FCNHNP /OR/ COMPADJ FCNHNP /THAN/ NP
(132,13) COMP2HNP <- DET COMPADJ FCNHNP /OR/ COMPADJ FCNHNP /THAN/ NP
(132,14) COMP2HNP <- COMP1HNP

(133,1)  COMP1HNP <- COMPADJ FCNHNP /THAN/ NP
(133,2)  COMP1HNP <- COMPADJ FCNHNP /THAN/ NP /HAVE/
(133,3)  COMP1HNP <- ARITHREL EXP FCNHNP
```

The semantic functions for these rules are currently UNDEFINED. The rules have been written with all the elements fully specified. When the semantics for them is carefully studied, I am sure that the rules can be compressed. The difference between COMP1HNP's and COMP2HNP's is that the former contains only one COMPADJ and the latter has a choice of two. All the contexts where COMP2HNP appears can have a COMP1HNP instead. (This is accomplished by rule (132,14).) Using the question format in rule (43,8), I will give one example of each of the COMPHNP's. Note that in many cases the final /HAVE/ is optional.

(132,1) & (132,2) Does 6 have more or less factors than 12 (has)?
(132,3) Does 6 have more or less than 3 factors?
(132,4) & (132,5) Does 6 have more factors or less factors than 12 (has)?
(132,6) & (132,7) Does 1/2 have a larger or smaller denominator than 1/3 (has)?
(132,8) & 132,9) Does 1/2 have a larger or a smaller denominator than 1/3 (has)?
(132,10) & (132,11) Does 1/2 have a larger denominator or a smaller denominator than 1/3 (has)?
(132,12) & (132,13) Does 2/3 have a larger denominator or larger numerator than 1/4 (has)?
(133,1) & (133,2) Does 6 have more even factors than 12 (has)?
(133,3) Does any prime number have more than 2 factors?

IV.49 HAVENPF-Rules

```
(43,7)    Q <- AUXIL CHOICELIST /HAVE/ HAVENPF
                                     (PICK ;4; ;2;)
(44,15)   Q1 <- INTER QU/I/HMNP /HAVE/ HAVENPF
                                     (APP ;4; ;2;)
(121,1)   HAVENPF <- /THE/ ORDADJ COMPADJ FCNHNP
                                     (HORDFCN ;2; ;3; ;4;)
(121,2)   HAVENPF <- /THE/ COMPADJ FCNHNP
                                     (HORDFCN 1 ;2; ;3;)
```

IV.50 HAVENP-Rules

```
(43,6)   Q <- AUXIL NP /HAVE/ HAVENP
                                 (S ;2; ;4;)
(43,23)  Q <- INTER2 /HAVE/ HAVENP PUNCHOICE CHOICELIST
                                 (PICK ;3; ;5;)
(44,16)  Q1 <- INTER QU/I/HMNP /HAVE/ HAVENP
                                 (I ;2; ;4;)
(44,24)  Q1 <- INTER EXP1 QU/I/HMNP /HAVE/ HAVENP
                                 (NMF ;2; (I ;3; :5;))
(44,42)  Q1 <- /HOWMANY/ QU/I/HMNP /HAVE/ HAVENP
                                 (LST (CARDINALITY (I ;2; ;4;)))

(122,1)  HAVENP <- EXTHNP                             ;1;
(122,2)  HAVENP <- UNVHNP                             ;1;
(122,3)  HAVENP <- EXP1 FCNHNP                        (EXPHNP ;1; ;2;)
(122,4)  HAVENP <- ANSHNP                             ;1;
(122,5)  HAVENP <- HAVENPCHOICE                       ;1;
(122,6)  HAVENP <- LISTOFHAVENP                       ;1;
(122,7)  HAVENP <- /NEITHER/ HAVENP /NOR// HAVENP     (NOR ;2; ;4;)
(122,8)  HAVENP <- /EITHER/ HAVENP /OR/ HAVENP        (U ;2; ;4;)

(129,1)  ANSHNP <- NP /AS/ ANSGVHNP                   (EXTHNP ;3; ;1;)
(129,2)  ANSHNP <- NP /AS/ ANSGVHNP /OR/ /AS/ ANSGVHNP
                                (CHL (EXTHNP ;3; ;1;) (EXTHNP ;6; ;1;))
(129,3)  ANSHNP <- NP /AS/ ANSGVHNP /AND/ /AS/ ANSGVHNP
                                (CHL (EXTHNP ;3; ;1;) (EXTHNP ;6; ;1;))
(129,4)  ANSHNP <- ANSGVHNP EXP                       (EXTHNP ;1; ;2;)
(129,5)  ANSHNP <- ANSGVHNP /OF/ EXP                  (EXTHNP ;1; ;3;)
(129,6)  ANSHNP <- /THE/ EXP1 FCNHNP EXP              UNDEFINED
(129,7)  ANSHNP <- EXP1 FCNHNP EXP                    UNDEFINED

(125,1)  EXTHNP <- FCNHNP                       (EXTHNP ;1; (STS UNIV))
(125,2)  EXTHNP <- FCNHNP /EXCEPT/ NP           (EXTHNP ;1; (C ;3;))
(125,3)  EXTHNP <- EXTHNPQU FCNHNP              (EXTHNP ;2; (STS UNIV))
(125,4)  EXTHNP <- EXTHNPQU FCNHNP /EXCEPT/ NP  (EXTHNP ;2; (C ;4;))

(124,1)  EXTHNPQU <- /ANY/
(124,2)  EXTHNPQU <- /SOME/
(124,3)  EXTHNPQU <- /A/

(127,1)  UNVHNP <- UNVHNPQU FCN  (UNVHNP (FCN ;2;) (STS UNIV))
(127,2)  UNVHNP <- UNVHNPQU FCN RESTRICT
                                 (UNVHNP (FCN ;2;) ;3;)
(127,3)  UNVHNP <- UNVHNPQU ADP FCN
                                 (UNVHNP (FCN ;3;) ;2;)
(127,4)  UNVHNP <- UNVHNPQU ADP FCN RESTRICT
                                 (UNVHNP (FCN ;3;) (I ;2; ;4;))
(127,5)  UNVHNP <- UNVHNPQU FCN /EXCEPT/ NP
                                 (UNVHNPXCT (FCN ;2;) (STS UNIV) ;4;)
```

```
(127,6)  UNVHNP <- UNVHNPQU FCN RESTRICT /EXCEPT/ NP
                                     (UNVHNPXCT (FCN ;2;) ;3; ;5;)
(127,7)  UNVHNP <- UNVHNPQU ADP FCN /EXCEPT/ NP
                                     (UNVHNPXCT (FCN ;3;) ;2; ;5;)
(127,8)  UNVHNP <- UNVHNPQU ADP FCN RESTRICT /EXCEPT/ NP
                                     (UNVHNPXCT (FCN ;3;) (I ;2; ;4;) ;6;)

(126,1)  UNVHNPQU <- /ALL/
(126,2)  UNVHNPQU <- /ONLY/
```

Appendix I

Examples of Questions and their Answers

The following are examples of questions accepted by the CONSTRUCT program. The format includes (a) the original sentence, (b) the dictionary classification of the sentence, (c) the semantic construction that, when evaluated, gives the meaning of the sentence, and (d) the evaluation of the sentence. "QUS" means that the sentence was a question, "TV" means "truth-value", "NIL" means "false", "T" means "true", "LST" means an explicit list of elements in a set, and "CHL" means that the question was a compound question with several answers.

```
(1a)  Is 2 a factor of 4?
(b)   link integer /a/ fcn /of/ integer ?
(c)   (QUS (S (LST 2) (APP (FCN @FACTOR) (LST 4))))
(d)   (QUS (TV T))
```

```
(2a)  Does 4 have a factor of 2?
(b)   aux integer /have/ /a/ fcn /of/ integer ?
(c)   (QUS (S (LST 4) (EXTHNP (FCN @FACTOR) (LST 2))))
(d)   (QUS (TV T))
```

```
(3a)  Are there any common factors of 4 and 12 that are greater than 4?
(b)   link /there/ /any/ 2fcn /of/ integer /and/
          integer relpron link arithrel integer ?
(c)   (QUS (EXIST (I (APP (FCN @COMMONFACTOR)
          (LST  (LST 4) (LST 12) )) (APP (FCN @GT) (LST 4)))))
(d)   (QUS (TV NIL))
```

```
(4a)  Are there any even prime numbers that are greater than 2?
(b)   link /there/ /any/ adj adj n relpron link arithrel integer ?
(c)   (QUS (EXIST (I (I (I (STS @EVEN) (STS @PRIME))
          (STS @NUMBER)) (APP (FCN @GT) (LST 2)))))
(d)   (QUS (TV NIL))
```

```
(5a)  Does the least common multiple of 4 and 5 come before
         the least common multiple of 4 and 12?
(b)   aux /the/ 2fcn /of/ integer /and/ integer /come/
         /before/ /the/ 2fcn /of/ integer /and/ integer ?
(c)   (QUS (S (APP (FCN @LCM) (LST  (LST 4) (LST 5) ))
         (BEFORE (APP (FCN @LCM) (LST  (LST 4) (LST 12) )))))
(d)   (QUS (TV NIL))
```

```
(6a)  Does 12 have any factors that are greater than 12?
(b)   aux integer /have/ /any/ fcn relpron link arithrel integer ?
(c)   (QUS (S (LST 12) (EXTHNP (FCNMK (FCN @FACTOR)
         (APP (FCN @GT) (LST 12))) (STS UNIV))))
(d)   (QUS (TV NIL))
```

```
(7a)  Are all the factors of 12 divisible by 2?
(b)   link /all/ /the/ fcn /of/ integer arithrel integer ?
(c)   (QUS (S (APP (FCN @FACTOR) (LST 12))
         (APP (FCN @DIVISIBLE) (LST 2))))
(d)   (QUS (TV NIL))
```

```
(8a)  Is the largest factor of 12 divisible by all
         the odd factors of 12?
(b)   link /the/ compadj fcn /of/ integer arithrel /all/
         /the/ adj fcn /of/ integer ?
(c)   (QUS (S (MAXF (FCN @GTF) (APP (FCN @FACTOR) (LST 12)))
         (APP (FCN @DIVISIBLE) (I (STS @ODD)
         (APP (FCN @FACTOR) (LST 12))))))
(d)   (QUS (TV T))
```

```
(9a)  Are 2 factors of 12 prime numbers that are odd?
(b)   link integer fcn /of/ integer adj n relpron link adj ?
(c)   (QUS (S (LST (CARDINALITY (I (APP (FCN @FACTOR) (LST 12))
         (I (I (STS @PRIME) (STS @NUMBER)) (STS @ODD)))))
         (APP (FCN @EQL) (LST 2))))
(d)   (QUS (TV NIL))
```

[Note: 1 is not the factor of any number according to the definition that we have implemented.]

```
(10a) Except for 4, what are the common factors of 4 and 12?
(b)   /except/ integer , inter link /the/ 2fcn /of/
         integer /and/ integer ?
(c)   (QUS (SD (APP (FCN @COMMONFACTOR)
         (LST  (LST 4) (LST 12) )) (LST 4)))
(d)   (QUS (LST 2))
```

```
(11a)  Which of the numbers 5, 16, 23, and 54 are even and
         divisible by 4?
(b)  inter /of/ /the/ apposn integer , integer , integer , /and/
         integer link adj /and/ arithrel integer ?
(c)  (QUS (I (I (STS @NUMBER)
         (CHL (LST 5) (LST 16)  (LST 23)  (LST 54) ))
         (I (STS @EVEN) (APP (FCN @DIVISIBLE) (LST 4)))))
(d)  (QUS (CHL (LST) (LST 16) (LST) (LST)))


(12a)  Which of the factors of 12 have only even factors that are
         greater than 4?
(b)  inter /of/ /the/ fcn /of/ integer /have/ /only/ adj fcn relpron
         link arithrel integer ?
(c)  (QUS (I (APP (FCN @FACTOR) (LST 12))
         (UNVHNP (FCN @FACTOR) (I (STS @EVEN)
         (APP (FCN @GT) (LST 4))))))
(d)  (QUS (LST))


(13a)  Is 3+2 less than, greater than, or equal to 2+3?
(b)  aux integer + integer arithrel , arithrel , /or/ arithrel
         integer + integer ?
(c)  (QUS (PICKFCN (CHL (FCN @LT) (FCN @GT) (FCN @EQL))
         (LST (ADDER 3 2) (ADDER 2 3))))
(d)  (QUS (LST (FCN EQL)))


(14a)  Which of these will be even:  2+2, 2+3, 3+2, 3+3 or 3+1?
(b)  inter /of/ /these/ mod /be/ adj : integer + integer ,
         integer + integer, integer  + integer /or/ integer + integer ?
(c)  (QUS (I (STS @EVEN)
         (CHL (LST (ADDER 2 2)) (LST (ADDER 2 3))
         (LST (ADDER 3 2))  (LST (ADDER 3 3))  (LST (ADDER 3 1)))))
(d)  (QUS (CHL (LST 4) (LST) (LST) (LST 6) (LST 4)))


(15a)  Which even number is a prime number -- 2 or 4?
(b)  inter adj n link /a/ adj n - - integer /or/ integer ?
(c)  (QUS (PICK (I (I (STS @EVEN) (STS @NUMBER))
         (I (STS @PRIME) (STS @NUMBER))) (CHL  (LST 2) (LST 4))))
(d)  (QUS (LST 2))


(16a)  What is the least common multiple of the
         product of 2 and 5 and the product of 2 and 6?
(b)  inter link /the/ 2fcn /of/ /the/ 2fcn /of/ integer
         /and/ integer /and/ /the/ 2fcn /of/ integer /and/ integer ?
```

(c) (QUS (S (APP (FCN @SUM) (LST (LST 5) (LST 2)))
(I (APP (FCN @LT) (APP (FCN @PRODUCT) (LST (LST 5)
(LST 2)))) (APP (FCN @GT) (APP (FCN @DIFFER)
(LST (LST 5) (LST 2))))))))
(d) (QUS (LST 60))

(17a) Is the sum of 5 and 2 less than the product of 5 and 2
but greater than the difference of 5 and 2?
(b) link /the/ 2fcn /of/ integer /and/ integer arithrel
/the/ 2fcn /of/ integer /and/ integer /and/ arithrel /the/
2fcn /of/ integer /and/ integer ?
(c) (QUS (S (APP (FCN @SUM) (LST (LST 5) (LST 2)))
(I (APP (FCN @LT) (APP (FCN @PRODUCT)
(LST (LST 5) (LST 2)))) (APP (FCN @GT)
(APP (FCN @DIFFER) (LST (LST 5) (LST 2)))))))
(d) (QUS (TV T))

(18a) Which of the numbers that are between 37 and 48 and are odd
does 86 have as factors?
(b) inter /of/ /the/ n relpron link /between/
integer /and/ integer /and/ link adj
aux integer /have/ /as/ fcn ?
(c) (QUS (I (I (STS @NUMBER)
(I (BETWEEN (LST 37) (LST 48)) (STS @ODD)))
(APP (FCN @FACTOR) (LST 86))))
(d) (QUS (LST 43))

(19a) Which of the factors of 36 are even and not between 1 and 36?
(b) inter /of/ /the/ fcn /of/ integer link adj /and/
/not/ /between/ integer /and/ integer ?
(c) (QUS (I (APP (FCN @FACTOR) (LST 36)) (I (STS @EVEN)
(C (BETWEEN (LST 1) (LST 36))))))
(d) (QUS (LST 36))

(20a) Which even number is a factor of 12 and a multiple of 3?
(b) inter adj n link /a/ fcn /of/ integer /and/ /a/ fcn /of/ integer ?
(c) (QUS (I (I (STS @EVEN) (STS @NUMBER))
(I (APP (FCN @FACTOR) (LST 12))
(APP (FCN @MULTIPLE) (LST 3)))))
(d) (QUS (LST 12 6))

(21a) How many factors of 4 are there that are also multiples of 4?
(b) /howmany/ fcn /of/ integer link /there/
relpron link fcn /of/ integer ?

```
(c)  (QUS (LST (CARDINALITY (I (APP (FCN @FACTOR) (LST 4))
         (APP (FCN @MULTIPLE) (LST 4))))))
(d)  (QUS (LST 1))


(22a)  Which number does 4 have both as a factor and as a multiple?
(b)  inter n aux integer /have/ /as/ /a/ fcn /and/ /as/ /a/ fcn ?
(c)  (QUS (I (STS @NUMBER) (I (APP (FCN @FACTOR) (LST 4))
         (APP (FCN @MULTIPLE) (LST 4)))))
(d)  (QUS (LST 4))


(23a)  How many even numbers between 3 and 50 have 7 as a factor?
(b)  /howmany/ adj n /between/ integer /and/ integer /have/
         integer /as/ /a/ fcn ?
(c)  (QUS (LST (CARDINALITY (I (I (STS @EVEN)
         (I (STS @NUMBER) (BETWEEN (LST 3) (LST 50))))
         (EXTHNP (FCN @FACTOR) (LST 7))))))
(d)  (QUS (LST 3))


(24a)  What are the even factors of 12 that are multiples of 4?
(b)  inter link /the/ adj fcn /of/ integer
         relpron link fcn /of/ integer ?
(c)  (QUS (I (I (STS @EVEN) (APP (FCN @FACTOR) (LST 12)))
         (APP (FCN @MULTIPLE) (LST 4))))
(d)  (QUS (LST 12 4))


(25a)  What is 7 divided into 56?
(b)  inter link integer /dividedinto/ integer ?
(c)  (QUS (LST (DIV 56 7)))
(d)  (QUS (LST (MXD 8 0)))


(26a)  What is 56 divided by 7?
(b)  inter link integer /dividedby/ integer ?
(c)  (QUS (LST (DIV 56 7)))
(d)  (QUS (LST (MXD 8 0)))


(27a)  How many even numbers are prime?
(b)  /howmany/ adj n link adj ?
(c)  (QUS (LST (CARDINALITY (I (I (STS @EVEN) (STS @NUMBER))
         (STS @PRIME)))))
(d)  (QUS (LST 1))
```

[Note: There is another semantically equivalent derivation in which 'prime' is parsed as a noun.]

(28a) Give the factors of 2, the factors of 3, and the factors of 4!
(b) /give/ /the/ fcn /of/ integer , /the/ fcn /of/ integer , /and/ /the/ fcn /of/ integer !
(c) (CMD (CHL (APP (FCN @FACTOR) (LST 2)) (APP (FCN @FACTOR) (LST 3)) (APP (FCN @FACTOR) (LST 4))))
(d) (CMD (CHL (LST 2) (LST 3) (LST 4 2)))

(29a) Give the numbers that are between 2 and 6 that are less than 4!
(b) /give/ /the/ n relpron link /between/ integer /and/ integer relpron link arithrel integer !
(c) (CMD (I (I (STS @NUMBER) (BETWEEN (LST 2) (LST 6))) (APP (FCN @LT) (LST 4))))
(d) (CMD (LST 3))

(30a) Give the factors of 12 and the factors of 15 that are prime numbers!
(b) /give/ /the/ fcn /of/ integer /and/ /the/ fcn /of/ integer relpron link adj n !
(c) (CMD (CHL (APP (FCN @FACTOR) (LST 12)) (I (APP (FCN @FACTOR) (LST 15)) (I (STS @PRIME) (STS @NUMBER)))))
(d) (CMD (CHL (LST 12 6 4 3 2) (LST 5 3)))

[Note: sentence (30) is genuinely ambiguous...]

(c') (CMD (I (CHL (APP (FCN @FACTOR) (LST 12)) (APP (FCN @FACTOR) (LST 15))) (I (STS @PRIME) (STS @NUMBER))))
(d') (CMD (CHL (LST 3 2) (LST 5 3)))

(31a) Is the largest factor of 5 even?
(b) link /the/ compadj fcn /of/ integer adj ?
(c) (QUS (S (MAXF (FCN @GTT) (APP (FCN @FACTOR) (LST 5))) (STS @EVEN)))
(d) (QUS (TV NIL))

(32a) Does 12 have a factor that is both even and prime?
(b) aux integer /have/ /a/ fcn relpron link adj /and/ adj ?
(c) (QUS (S (LST 12) (EXTHNP (FCNMK (FCN @FACTOR) (I (STS @EVEN) (STS @PRIME))) (STS UNIV))))
(d) (QUS (TV T))

(33a) Is the largest common factor of 20 and 24 odd or even?
(b) link /the/ compadj 2fcn /of/ integer /and/ integer adj /or/ adj ?
(c) (QUS (S (MAXF (FCN @GTT) (APP (FCN @COMMONFACTOR) (LST (LST 20) (LST 24)))) (CHL (STS @ODD) (STS @EVEN))))
(d) (QUS (CHL (TV NIL) (TV T)))

(34a) Will the product of 2 and 4 come before the sum of 2 and 4?
(b) mod /the/ 2fcn /of/ integer /and/ integer /come/ /before/ /the/ 2fcn /of/ integer /and/ integer ?
(c) (QUS (S (APP (FCN @PRODUCT) (LST (LST 2) (LST 4))) (BEFORE (APP (FCN @SUM) (LST (LST 2) (LST 4))))))
(d) (QUS (TV NIL))

(35a) Is 4 a common multiple of 2 and 4?
(b) link integer /a/ 2fcn /of/ integer /and/ integer ?
(c) (QUS (S (LST 4) (APP (FCN @COMMONMULTIPLE) (LST (LST 2) (LST 4)))))
(d) (QUS (TV T))

(36a) Which of these will be even: the sum of 5 and 2, the difference of 5 and 2 , or the product of 5 and 2?
(b) inter /of/ /these/ mod /be/ adj : /the/ 2fcn /of/ integer /and/ integer , /the/ 2fcn /of/ integer /and/ integer , /or/ /the/ 2fcn /of/ integer /and/ integer ?
(c) (QUS (I (STS @EVEN) (CHL (APP (FCN @SUM) (LST (LST 5) (LST 2))) (APP (FCN @DIFFER) (LST (LST 5) (LST 2))) (APP (FCN @PRODUCT) (LST (LST 5) (LST 2))))))
(d) (QUS (CHL (LST) (LST) (LST 10)))

(37a) How many prime numbers are there between 10 and 20?
(b) /howmany/ adj n link /there/ /between/ integer /and/ integer ?
(c) (QUS (LST (CARDINALITY (I (I (STS @PRIME) (STS @NUMBER)) (BETWEEN (LST 10) (LST 20))))))
(d) (QUS (LST 4))

(38a) How many numbers between 5 and 10 are odd numbers?
(b) /howmany/ n /between/ integer /and/ integer link adj n ?
(c) (QUS (LST (CARDINALITY (I (I (STS @NUMBER) (BETWEEN (LST 5) (LST 10))) (I (STS @ODD) (STS @NUMBER))))))
(d) (QUS (LST 2))

(39a) What does 3 + 5 equal?
(b) inter aux integer + integer = ?
(c) (QUS (LST (ADDER 3 5)))
(d) (QUS (LST 8))

(40a) Which 4 numbers between 10 and 20 are prime numbers?
(b) inter integer n /between/ integer /and/ integer link adj n ?

(c) (QUS (NMF 4 (I (I (STS @NUMBER) (BETWEEN (LST 10) (LST 20))) (I (STS @PRIME) (STS @NUMBER)))))
(d) (QUS (LST 19 17 13 11))

(41a) How many even factors that are between 10 and 50 does 100 have?
(b) /howmany/ adj fcn relpron link /between/ integer /and/ integer aux integer /have/ ?
(c) (QUS (LST (CARDINALITY (APP (FCNMK (FCN @FACTOR) (I (STS @EVEN) (BETWEEN (LST 10) (LST 50)))) (LST 100)))))
(d) (QUS (LST 1))

(42a) Does 12 have 6 as a factor or as a multiple?
(b) aux integer /have/ integer /as/ /a/ fcn /or/ /as/ /a/ fcn ?
(c) (QUS (S (LST 12) (CHL (EXTHNP (FCN @FACTOR) (LST 6)) (EXTHNP (FCN @MULTIPLE) (LST 6)))))
(d) (QUS (CHL (TV T) (TV NIL)))

(43a) Does 12 have 12 as a factor and also as a multiple?
(b) aux integer /have/ integer /as/ /a/ fcn /and/ /as/ /a/ fcn ?
(c) (QUS (S (LST 12) (CHL (EXTHNP (FCN @FACTOR) (LST 12)) (EXTHNP (FCN @MULTIPLE) (LST 12)))))
(d) (QUS (CHL (TV T) (TV T)))

(44a) Does 6 have any factors that are also factors of 3?
(b) aux integer /have/ /any/ fcn relpron link fcn /of/ integer ?
(c) (QUS (S (LST 6) (EXTHNP (FCNMK (FCN @FACTOR) (APP (FCN @FACTOR) (LST 3))) (STS UNIV))))
(d) (QUS (TV T))

(45a) Which factor of 6 is also a factor of 3?
(b) inter fcn /of/ integer link /a/ fcn /of/ integer ?
(c) (QUS (I (APP (FCN @FACTOR) (LST 6)) (APP (FCN @FACTOR) (LST 3))))
(d) (QUS (LST 3))

(46a) Does 6 have any factors that are also multiples of 6?
(b) aux integer /have/ /any/ fcn relpron link fcn /of/ integer ?
(c) (QUS (S (LST 6) (EXTHNP (FCNMK (FCN @FACTOR) (APP (FCN @MULTIPLE) (LST 6))) (STS UNIV))))
(d) (QUS (TV T))

(47a) Are there any factors of 6 that are also multiples of 6?
(b) link /there/ /any/ fcn /of/ integer relpron link fcn /of/ integer ?

(c) (QUS (EXIST (I (APP (FCN @FACTOR) (LST 6)) (APP (FCN @MULTIPLE)
(LST 6)))))
(d) (QUS (TV T))

(48a) Does 10 have any even factors that are between 2 and 10?
(b) aux integer /have/ /any/ adj fcn relpron link /between/
integer /and/ integer ?
(c) (QUS (S (LST 10) (EXTHNP (FCNMK (FCN @FACTOR) (I (STS @EVEN)
(BETWEEN (LST 2) (LST 10)))) (STS UNIV))))
(d) (QUS (TV NIL))

(49a) Is the factor of 10 that is between 2 and 10 odd or even?
(b) link /the/ fcn /of/ integer relpron link /between/
integer /and/ integer adj /or/ adj ?
(c) (QUS (S (I (APP (FCN @FACTOR) (LST 10))
(BETWEEN (LST 2) (LST 10))) (CHL (STS @ODD) (STS @EVEN))))
(d) (QUS (CHL (TV T) (TV NIL)))

(50a) Does 12 have any factors that are greater than 6 that are odd?
(b) aux integer /have/ /any/ fcn relpron link arithrel integer relpron
link adj ?
(c) (QUS (S (LST 12) (EXTHNP (FCNMK (FCN @FACTOR) (I (APP (FCN @GT)
(LST 6)) (STS @ODD))) (STS UNIV))))
(d) (QUS (TV NIL))

Index

adjectives 29, 43
agreement 17
ambiguity 22, 64, 111
arithmetic expressions 82, 85
arithmetic relations 30

clarity 14
commands 107

CONSTRUCT 2, 51

constructive sets 5
control structure 13, 45

data structures 35
declaratives 109

DICTIONARY 2

Elementary mathematics 5

evaluation techniques 35
evaluator 2
existential quantifier 47
extendability 32

flexibility 32

grammar 2, 22, 59
grammar writing 61

interaction 14

lexical categories 53

lists 78, 98

measurements 88
models of semantics 27
multiple categories 53
multiple choice format 127

noun phrases 80
nouns 29

pattern recognition 28, 58
prepositional phrases 16
prepositions 91
primitive semantic functions 11
probabilistic grammars 66
programming languages 34

question-answering system 2

recursive evaluation 13
restructuring 44

scanner 2, 53
semantic categories 17
semantic construction 2, 12, 72
semantic function 2
semantics 10, 14, 17, 22, 60
subject matter 4
syntax 10, 14, 17, 22, 28

theorem prover 37
transformational semantic functions 6, 13
transformations 38

TRANSL 2, 57

unbounded branching 43, 75

UNITs 87

universal quantifier 49

vocabulary 8

References

1. Black, Fischer, A deductive question-answering system, Semantic Information Processing, Marvin Minsky (Ed.), MIT Press, Cambridge, Massachusetts, 1968, pp. 354-401.

2. Bobrow, Daniel G., Natural language input for a computer problem solving system, Semantic Information Processing, Marvin Minsky (Ed.), MIT Press, Cambridge, Massachusetts, 1968, pp. 146-226.

3. Chomsky, N., A transformational approach to syntax, The Structure of Language, J.A. Fodor and J.J. Katz (Eds.), Prentice-Hall, Englewood Cliffs, New Jersey, 1964.

4. Colby, Kenneth Mark, and Enea, Horace, Idiolectic language analysis for understanding doctor-patient dialogues, Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford, Calif., (1973), pp. 278-284.

5. Craig, J.A., Berezner, S.C., Carney, H.C., and Longyear, C.R., DEACON: Direct English Access and Control, AFIPS Conference Proceedings, 29, (1966), pp. 365-380.

6. Fillmore, Charles J., The case for case, Universals in Linguistic Theory, E. Bach and A. Harms (Eds.), Holt, Rinehart, and Winston, New York, 1968, p. 1-88.

7. Gries, David, Compiler Construction for Digital Computers, John Wiley and Sons, New York, 1971.

8. Katz, Jerrold J., Recent issues in semantic theory, Foundations of Language 3, (1968), pp. 124-194.

9. Lindsay, Robert K., Inferential memory as the basis of machines which understand natural language, Computers and Thought, Edward A. Feigenbaum and Julian Feldman (Eds.), McGraw-Hill, New York, 1963, pp. 217-233.

10. Minsky, Marvin, Introduction to *Semantic Information Processing*, MIT Press, Cambridge, Massachusetts, 1968.

11. Montague, Richard, English as a formal language, *Linguaggi nella Societa e nella Tecnica* (Language in Society and the Technical World), Milan, 1970.

12. Palme, J., Making computers understand natural language, *Artificial Intelligence and Heuristic Programming*, N. Findler and B. eltzer (Eds.), Edinburgh University Press, 1971, pp. 199-244.

13. Postal, Paul M., Limitations of phrase structure grammars, *The Structure of Language*, J. A. Fodor and J. J. Katz (Eds.), Prentice-Hall, Englewood Cliffs, New Jersey, 1964, pp. 137-151.

14. Quillian, M. Ross, Semantic memory, *Semantic Information Processing*, Marvin Minsky (Ed.), MIT Press, Cambridge, Cambridge, Massachusetts, 1968, pp. 227-270.

15. Quillian, M. Ross, The teachable language comprehender: A simulation program and theory of language, *Communications of the Association for Computing Machinery*, Vol. 12 (1969), No. 8, pp. 459-475.

16. Raphael, Bertram, SIR: A computer program for semantic information retrieval, *Semantic Information Processing*, Marvin Minsky (Ed.), MIT Press, Cambridge, Massachusetts, 1968, pp. 33-145.

17. Rawson, Freeman L. III, Set-theoretical semantics for elementary mathematical language, Doctoral Dissertation, Stanford University, 1973. Also Technical Report 220, Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

18. Sager, Naomi, Syntactic formatting of science information, *AFIPS Conference Proceedings*, 41, (1972), pp. 791-800.

19. Sandewall, Eric, Formal methods in the design of question-answering systems, Artificial Intelligence 2, (1971), pp. 129-145.

20. Schank, Roger C. and Tesler, Lawrence G., A conceptual parser for natural language, Proceedings of the International Joint Conference on Artificial Intelligence, Washington, D.C., (1969), pp. 569-578.

21. Smith, Robert L. Jr., The syntax and semantics of ERICA. Doctoral dissertation, Stanford University, 1972. Also Technical Report 185, Institute for Mathematical Studies in the Social Sciences, Stanford University, 1972.

22. Smith, Robert L. Jr., forthcoming.

23. Suppes, Patrick, Sets and Numbers, L. W. Singer Company, New York, 1969.

24. Thompson, Frederick B., English for the Computer, AFIPS Conference Proceedings, 29, (1966), pp. 349-356.

25. Weizenbaum, Joseph, ELIZA -- A computer program for the study of natural language communication between man and machine, Communications of the Association for Computing Machinery, Vol. 9(1966), No. 1, pp. 36-45.

26. Winograd, Terry, Procedures as a representation for data in a computer program for understanding natural language, Doctoral dissertation, Massachusetts Institute of Technology, 1971.

27. Woods, W. A., Procedural semantics for a question-answering machine, AFIPS Conference Proceedings, (1968), pp. 457-471.

28. Woods, W.A., Transition network grammars for natural language analysis, Communications of the Association for Computing Machinery, Vol. 13 (1970), No. 10, pp. 591-606.

(Continued from inside front cover)

165 L. J. Hubert. A formal model for the perceptual processing of geometric configurations. February 19, 1971. (A statistical method for investigating the perceptual confusions among geometric configurations. *Journal of Mathematical Psychology*, 1972, *9*, 389-403.)

166 J. F. Juola, I. S. Fischler, C. T. Wood, and R. C. Atkinson. Recognition time for information stored in long-term memory. (*Perception and Psychophysics*, 1971, *10*, 8-14.)

167 R. L. Klatzky and R. C. Atkinson. Specialization of the cerebral hemispheres in scanning for information in short-term memory. (*Perception and Psychophysics*, 1971, *10*, 335-338.)

168 J. D. Fletcher and R. C. Atkinson. An evaluation of the Stanford CAI program in initial reading (grades K through 3). March 12, 1971. (Evaluation of the Stanford CAI program in initial reading. *Journal of Educational Psychology*, 1972, *63*, 597-602.)

169 J. F. Juola and R. C. Atkinson. Memory scanning for words versus categories. (*Journal of Verbal Learning and Verbal Behavior*, 1971, *10*, 522-527.)

170 I. S. Fischler and J. F. Juola. Effects of repeated tests on recognition time for information in long-term memory. (*Journal of Experimental Psychology*, 1971, *91*, 54-58.)

171 P. Suppes. Semantics of context-free fragments of natural languages. March 30, 1971. (In K. J. J. Hintikka, J. M. E. Moravcsik, and P. Suppes (Eds.), *Approaches to natural language*. Dordrecht: Reidel, 1973. Pp. 221-242.)

172 J. Friend. INSTRUCT coders' manual. May 1, 1971.

173 R. C. Atkinson and R. M. Shiffrin. The control processes of short-term memory. April 19, 1971. (The control of short-term memory. *Scientific American*, 1971, *224*, 82-90.)

174 P. Suppes. Computer-assisted instruction at Stanford. May 19, 1971. (In *Man and computer*. Proceedings of international conference, Bordeaux, 1970. Basel: Karger, 1972. Pp. 298-330.)

175 D. Jamison, J. D. Fletcher, P. Suppes, and R. C. Atkinson. Cost and performance of computer-assisted instruction for education of disadvantaged children. July, 1971.

176 J. Offir. Some mathematical models of individual differences in learning and performance. June 28, 1971. (Stochastic learning models with distribution of parameters. *Journal of Mathematical Psychology*, 1972, *9*(4),)

177 R. C. Atkinson and J. F. Juola. Factors influencing speed and accuracy of word recognition. August 12, 1971. (In S. Kornblum (Ed.), *Attention and performance* IV. New York: Academic Press, 1973.)

178 P. Suppes, A. Goldberg, G. Kanz, B. Searle, and C. Stauffer. Teacher's handbook for CAI courses. September 1, 1971.

179 A. Goldberg. A generalized instructional system for elementary mathematical logic. October 11, 1971.

180 M. Jerman. Instruction in problem solving and an analysis of structural variables that contribute to problem-solving difficulty. November 12, 1971. (Individualized instruction in problem solving in elementary mathematics. *Journal for Research in Mathematics Education*, 1973, *4*, 6-19.)

181 P. Suppes. On the grammar and model-theoretic semantics of children's noun phrases. November 29, 1971.

182 G. Kreisel. Five notes on the application of proof theory to computer science. December 10, 1971.

183 J. M. Moloney. An investigation of college student performance on a logic curriculum in a computer-assisted instruction setting. January 28, 1972.

184 J. E. Friend, J. D. Fletcher, and R. C. Atkinson. Student performance in computer-assisted instruction in programming. May 10, 1972.

185 R. L. Smith, Jr. The syntax and semantics of ERICA. June 14, 1972.

186 A. Goldberg and P. Suppes. A computer-assisted instruction program for exercises on finding axioms. June 23, 1972. (*Educational Studies in Mathematics*, 1972, *4*, 429-449.)

187 R. C. Atkinson. Ingredients for a theory of instruction. June 26, 1972. (*American Psychologist*, 1972, *27*, 921-931.)

188 J. D. Bonvillian and V. R. Charrow. Psycholinguistic implications of deafness: A review. July 14, 1972.

189 P. Arabie and S. A. Boorman. Multidimensional scaling of measures of distance between partitions. July 26, 1972. (*Journal of Mathematical Psychology*, 1973, *10*,)

190 J. Ball and D. Jamison. Computer-assisted instruction for dispersed populations: System cost models. September 15, 1972. (*Instructional Science*, 1973, *1*, 469-501.)

191 W. R. Sanders and J. R. Ball. Logic documentation standard for the Institute for Mathematical Studies in the Social Sciences. October 4, 1972.

192 M. T. Kane. Variability in the proof behavior of college students in a CAI course in logic as a function of problem characteristics. October 6, 1972.

193 P. Suppes. Facts and fantasies of education. October 18, 1972. (In M. C. Wittrock (Ed.), *Changing education: Alternatives from educational research*. Englewood Cliffs, N. J.: Prentice-Hall, 1973. Pp. 6-45.)

194 R. C. Atkinson and J. F. Juola. Search and decision processes in recognition memory. October 27, 1972.

195 P. Suppes, R. Smith, and M. Léveillé. The French syntax and semantics of PHILIPPE, part 1: Noun phrases. November 3, 1972.

196 D. Jamison, P. Suppes, and S. Wells. The effectiveness of alternative instructional methods: A survey. November, 1972.

197 P. Suppes. A survey of cognition in handicapped children. December 29, 1972.

198 B. Searle, P. Lorton, Jr., A. Goldberg, P. Suppes, N. Ledet, and C. Jones. Computer-assisted instruction program: Tennessee State University. February 14, 1973.

199 D. R. Levine. Computer-based analytic grading for German grammar instruction. March 16, 1973.

200 P. Suppes, J. D. Fletcher, M. Zanotti, P. V. Lorton, Jr., and B. W. Searle. Evaluation of computer-assisted instruction in elementary mathematics for hearing-impaired students. March 17, 1973.

201 G. A. Huff. Geometry and formal linguistics. April 27, 1973.

202 C. Jensema. Useful techniques for applying latent trait mental-test theory. May 9, 1973.

203 A. Goldberg. Computer-assisted instruction: The application of theorem-proving to adaptive response analysis. May 25, 1973.

204 R. C. Atkinson, D. J. Herrmann, and K. T. Wescourt. Search processes in recognition memory. June 8, 1973.

205 J. Van Campen. A computer-based introduction to the morphology of Old Church Slavonic. June 18, 1973.

206 R. B. Kimball. Self-optimizing computer-assisted tutoring: Theory and practice. June 25, 1973.

207 R. C. Atkinson, J. D. Fletcher, E. J. Lindsay, J. O. Campbell, and A. Barr. Computer-assisted instruction in initial reading. July 9, 1973.

208 V. R. Charrow and J. D. Fletcher. English as the second language of deaf students. July 20, 1973.

209 J. A. Paulson. An evaluation of instructional strategies in a simple learning situation. July 30, 1973.

210 N. Martin. Convergence properties of a class of probabilistic adaptive schemes called sequential reproductive plans. July 31, 1973.

(Continued from inside back cover)

211 J. Friend. Computer-assisted instruction in programming: A curriculum description. July 31, 1973.

212 S. A. Weyer. Fingerspelling by computer. August 17, 1973.

213 B. W. Searle, P. Lorton, Jr., and P. Suppes. Structural variables affecting CAI performance on arithmetic word problems of disadvantaged and deaf students. September 4, 1973.